N:GC33-0005-3
File No. S360/S370-29

DOS
PL/I 最適化コンパイラー：言語解説書

Program Product

プログラム番号 5736-PL1
(このプロダクトは，複合パッケージ
5736-PL3の一部としても配布される)

IBM

本書は，DOS PL／I 最適化コンパイラーの第1版，リリース4，
修正レベル0（プログラム番号 5736－PL1），ならびに，それ
以降に適応します。

原 典：GC33-0005-3
DOS
PL／I Optimizing Compiler
Language Reference Manual

翻訳担当：営業資料センター 資料開発

第1刷．1974．11

まえがき

本書は，ＰＬ／Ｉのプログラマーに参照資料としてご使用いただくよう，編さんされたものです。本書は，初歩的な入門書ではなく，すでにＰＬ／Ｉ言語を習得され，参照のもととなるものを必要とされている方を対象にしています。

本書は，２部にわかれています。第Ｉ部では，この言語の概念を解説します。第Ⅱ部では，細かい規則や統語法について説明します。

実施面に関する情報も一応は扱いますが，各々の実施環境について，すべて網羅してあるわけではありません。一般的に云って，本書には，DOS PL/I 最適化コンパイラーで処理される予定のプログラムを書くのに必要な情報が含まれているだけで，プログラムを実行するのに必要なすべての情報が含まれているわけではありません。プログラムの実行に関する詳しいことは，このコンパイラーのプログラマーの手引をごらんください。

このコンパイラーで処理されたプログラムを実行するためには，サブルーチン・ライブラリーが必要です。これらのサブルーチン・ライブラリーは，次のようなＩＢＭシステム／３６０のプログラム・プロダクトに入っています。

DOS PL/I 常駐ライブラリー，
プログラム・プロダクト　５７３６－ＬＭ４

DOS PL/I 非常駐ライブラリー，
プログラム・プロダクト　５７３６－ＬＭ５

本書の用い方

本書は，ＰＬ／Ｉプログラマーが用いる参照資料として作られたものです。使用者の必要性に応じて，詳細にも，また簡潔にも，参照する事項を迅速に見つけることができるよう，本書は２部にわけて，資料を提示してあります。

第Ｉ部「ＰＬ／Ｉの概念」は，ＰＬ／Ｉ言語の様々な機能とそれらの相互関係を解説する説明文と例題から構成されています。互いに参照し合うことを避けるため，また，それぞれの章をその主題に関する完全な参照資料として独立したものにするため，ある章に出てきた情報が別の章にも繰返し出てくることもあります。しかし，第Ｉ部は，全体を順々に読んでゆくこともできます。

第Ⅱ部「規則と構文法」は，特定の情報を迅速に参照するためのものです。ここには，相互の関係についての情報はごくわずかしか含まれていませんが，特定の疑問に対する回答が早く得られるように構成してあります。第Ⅱ部は，参照に都合がよいということを主眼として構成してあり，順に読む上で都合のよい構成にはなっていません。

たとえば，プログラマーが，プログラムの中で使う様々なステートメントの相互関係についての情報を知りたいときは，第Ｉ部第５章の「ステートメントの分類」を読めばよいのですが，特定のステートメントの使用法，その効果，許されるオプション，書き方の形式などに関するすべての規則を知りたいときは，第Ⅱ部Ｊ節「ステートメント」をみる方がよいのです。

同様に，データ変換の概念の解説については，第Ｉ部第４章の「式とデータ変換」を読み，特定のタイプの変換が行なわれたときの正確な結果を調べたいときは，第Ⅱ部Ｆ節の「データの変換と式の値の計算」をお読みください。

ＰＬ／Ｉの要素を記述するために本書で使用している統語言語は，第Ⅱ部Ａ節の「ＰＬ／Ｉの構文表記法」に説明してあります。

必読の資料

プログラムのコンパイル，連係編集，および実行をする際に必要な情報について，読者は，次の資料をよく読んでおかなければなりません。

DOS PL/I Optimizing Compiler: Programmer's Guide, SC33-0008

推せん資料

次にあげる資料の中で取りあげられている項目は，コンパイラーの能力，最適化の機能，オペレーティング・システムの環境，およびＤＯＳ ＰＬ／Ｉ最適化コンパイラーで実施される言語とＤＯＳ ＰＬ／Ｉ Ｄコンパイラーで実施される言語との比較，などです。

DOS PL/I Optimizing Compiler: General Information, GC33-0004

DOS PL/I Optimizing Compiler: Execution Logic, SC33-0019

資料の入手方法

資料の入手方法は，ユース・キー（注文番号の最初の文字）によって表示されます。本書の中で参照されている資料のユース・キーは，以下の通りです。

G － 一般向き： ＩＢＭシステム，プロダクトおよびサービスの使用者は，ごく普通の使用量の範囲内であれば無料で入手することができます。また，この資料は，ＩＢＭの営業所を通じて，どなたでも購入することができます。

S － 販売用： ＩＢＭの営業所を通じて，どなたでも購入することができます。

目　次

図

第Ｉ部：PL／I の概念

第１章：　ＰＬ／Ｉの基本的な特徴

ＰＬ／Ｉ言語の様々な機能を調べてみると，ＰＬ／Ｉが積木方式になっており，そのために，様々な要件に合わせてサブセットを容易に選択できるようになっていることがわかる。このような積木方式は，ＰＬ／Ｉの最も重要な特徴の１つである。

この章では，この言語の全般的な概要を知るために，数々の基本的な特徴のほゞすべてについて，簡単に説明する。各々の特徴の詳細は，後の章で解説する。

機械からの独立性

使用する機械を全く考慮しないでプログラムを書くことのできる言語はないが，それでもＰＬ／Ｉは，一般によく使われている他の言語にくらべれば，機械に依存する度合が非常に少ないといえる。独立性を得るために用いられている方法は，言語の中では制約の形となって表われている。最も顕著な例として，異なる性質をもつデータは通常同じ記憶域を共有することができない，ということがあげられる。浮動小数点数と所定の桁数の英字を等しいものにするということは，これらのデータ項目の表現について特別な想定をしていることになり，これは，すべての機械について正しいとはいえないからである。

機械からの独立性を得るために払われる代償が，ときには非常に高価なものになることもある。効率をよくするという観点から，UNSPEC 組込み関数やレコード方式のデータ転送などの特定の機能では，ある程度機械に依存することを許している。

プログラムの構造

ＰＬ／Ｉのプログラムは，プロシージャーと呼ばれる１つまたは２つ以上のステートメントのブロックからなっている。プロシージャーは，サブルーチンと考えてもよい。プロシージャーは他のプロシージャーを呼び出すことができるが，このような場合のプロシージャーまたはサブルーチンは，別々にコンパイルしてもよいし，あるいは呼出し側のプロシージャーの中にネストしておき，いっしょにコンパイルしてもよい。各プロシージャーごとに，名前を定義する宣言や記憶域の割振りを制御する宣言を含めることができる。

ごく初歩的な段階をのぞいて，ＰＬ／Ｉを正しく理解する上で，プロシージャーの使い方を定義する規則，プロシージャー間の連絡，名前の意味，および記憶域の割振りは，基本的で重要な事項である。これらの規則を応用することによってプログラマーは，サブルーチン間の相互関係の度合を，かなり制御することができる。これらの規則は，柔軟性のある連絡や記憶域割振りを許しているが，それと同時に名前の定義や記憶域の割振りを１つのプロシージャー内で個有のものにしておくことも可能にしている。

ＰＬ／Ｉでは，サブルーチンを自己内蔵的にしておく度合を決定する自由がプログラマーに与えられているので，プログラマーは，他の環境でも自由に使用できるプログラムを書くことができるが，一方では，相互関係が必要なときはプロシージャーの中に相互作用の要素を盛り込むこともできる。

データのタイプとデータの記述

ＰＬ／Ｉが広い範囲の適用業務に使用できる最も大きな理由の１つは，表現，取り扱いの可能なデータのタイプが多いということである。ＰＬ／Ｉは，算術データ，ストリング・データ（ビットと文字），およびラベルなどのプログラム制御データを取り扱う。算術データの表現方法には様々なものがあり，２進数または１０進数，固定小数または浮動小数，実数または複素数のいずれでもよく，また，その精度も指定することができる。

ＰＬ／Ｉが行なう演算には，算術演算，比較演算，ビット・ストリングの論理操作，文字ストリングの組み立て，走査，分割の操作と機能がある。

コンパイラーは，プログラムの中で使用されているすべての名前と，その名前に関連する属性のセット全体を判定することができなければならない。これらの属性は，プログラマーがDECLARE ステートメントを用いて明示的に指定することもできるし，コンパイラーが文脈から属性のすべてまたは一部を決定することもできる。また，属性の一部または全体が省略時解釈によって想定されることもある。プログラマーは，どの属性に省略時解釈を適用するか指定することもできるし，あるいはそれらをコンパイラーに決めさせることもできる。

省略時解釈

ＰＬ／Ｉの重要な機能の１つに，省略時解釈という概念がある。ある１

つの名前に結びつけられる属性のすべて，またはある1つのステートメントで許されるオプションのすべてをプログラマーが指定しなかった場合は，コンパイラーが属性またはオプションを割り当てる。この省略時解釈の働きから，次のような2つの大きな結果が得られる。第一に，宣言の数やその他のプログラムを書く手間が減る。第二に，プログラマーは，その中に入っているすべての選択項目を知らなくても，場合によっては選択項目の存在さえも知らなくても，この言語のサブセットを教えたり，使用したりすることができる。

コンパイラーが想定する省略時解釈の属性は，PL／I 言語の標準的な省略時解釈の属性と，そのコンパイラーで実施される精度の省略時解釈である。しかし，プログラマーは，DEFAULT ステートメントを用いて，これらを打ち消すこともできる。

任意選択によって，コンパイラーは，PL／I 原始プログラムの中で使用されている識別名が記載されている属性のリスティング，および明示宣言または文脈宣言によって指定された属性，または省略時解釈が適用された結果指定された属性の完全なリストを作成する。プログラマーは，これらのリスティングを使用して，これらの属性が意図したものと矛盾していないかチェックすることができる。

記憶域の割振り

PL／I は，記憶域の割振りの融通性という点で，他の言語よりはるかに優れている。動的に記憶域を割振ることは，アセンブラー言語のプログラマーが自分自身で処理するとすれば相当難しい手法であるが，PL／I では，これが自動的に行なわれる。記憶域クラスには，AUTOMATIC，STATIC，CONTROLLED，BASED の4種類がある。通常，PL／I の記憶域クラスに関する省略時解釈は，AUTOMATICである。このクラスの記憶域は，変数が宣言されているブロックが活用化されたときにいつも割り振られる。また，この時点で，配列の限界とストリングの長さが計算される。AUTOMATIC記憶域は，その記憶域が割り振られているブロックから制御が移ったときはつねに，解放され，再び使用することができるようになる。

記憶域はSTATIC と宣言することもできるが，この場合は，プログラムがロードされるときに記憶域が割り振られる。CONTROLLEDと宣言したときは，ブロックの呼出しとは無関係に，プログラマーがALLOCATEおよびFREEステートメントを用いて明示的に制御する。BASED と宣言した場合は，プログラマーはさらに高度の制御をすることができる。

このように，幾種類もの記憶域クラスがあるので，プログラマーは，それぞれの適用業務に応じて，スピード，記憶域スペース，またはプログラミングの経済性などを，自分自身で判断することができる。特定の機能を実行する上でかかるコストは，実施環境によって異なるが，一般に，記憶域の割振りが動的になるほど，実行時間が大きくなる，ということが云える。

式

PL／I では，計算は式を使って指定する。PL／I の式は，基本的な数式と同じ意味をもっている。たとえば，

A + B * C

上の式は，Bの値にCの値を掛け，その結果にAの値を加えることを指定している。PL／I は，式の中で使用できるデータの種類にほとんど制約を加えていない。たとえば，あまりあり得ることではないが，Aが浮動小数点数，Bが固定小数点数，Cが文字ストリングということも可能である。

このようないくつかのタイプのデータが混ざっている式を指定すると，意味のある演算ができるようにオペランドが変換される。しかし，このような場合は，変換されたデータが元の値と同じにならないこともあるので，変換の規則を充分考慮しなければならない。また，変換が行なわれれば，それだけ実行時間が増すのは当然である。

式の計算結果は，割当てステートメントによって変数に割り当てられる。次にあげるのは，割当てステートメントの例である。

X = A + B * C;

この意味は，右辺の式を計算して，その計算結果をXに貯えるということである。Xの属性が式の計算結果の属性と違っている場合は，再び変換が行なわれる。

データの集まり

PL／I には，データの集まり，つまりデータの集合体について記述したり，操作を行なったりするための方法がいくつか用意されている。配列は，全部が同じタイプのデータ要素を集めたもので，1次元または数次元にわたるリストまたはテーブルの形になっている。構造は，上下関係をもたせたデータの集まりで，すべてが同じタイプのものである必要はない。上下関係にある各レベルの中に，さらに低いレベルをもつ別の構造が含まれていてもよい。別の構造をその中に含まない項目は，基本データ項目または配列を表わしていなければならない。

配列の要素が構造であってもよい。同様に，どのレベルの構造も配列であってよい。配列，構造，あるいは配列または構造の一部に対して演算を指定することもできる。たとえば，

A = B + C；

この割当てステートメントの中で，A，B，およびCが，配列または構造であってよい。

入力と出力

入出力に関する機能がいくつも用意されているので，ユーザーは，簡便さ，機械への依存度，効率などを考え合わせて，その要素の中から選択することができる。PL／Iの入出力には，大きくわけて，ストリーム式とレコード式の2種類がある。

ストリーム式入出力は，ほぼ完全に機械から独立した形で行なわれる。入力の際には，連続する文字の流れと考えられるものから，データ項目が1つずつ選択されて，内部形式に変換され，リストの中で指定された変数に割り当てられる。同様に，出力の際には，データ項目は1つずつ外部文字形式に変換されて，概念的には連続する文字の流れと考えられるものに加えられてゆく。プログラマーは，データ項目の編集の仕方や，流れの中からそれらを選択したり，流れに入れたりする方法に対して加えられる制御の様々のレベルを，ストリーム式入出力のクラスの中から選択することができる。

印刷の場合，出力の流れは行とページに分けられていると考えることができる。出力ストリーム・ファイルは，行の大きさとページの大きさを指定して，印刷ファイルとして宣言することができる。ページの終りを検出したり，行またはページの始まりを指定する能力が，プログラマーに与えられている。これらの機能をサブルーチンの中で使用して，特定の客先または適用業務に適した報告書作成システムにまで発展させてゆくことも可能である。

レコード式入出力は，機械に依存している。この方式では，レコードと呼ばれるデータの集まりを取り扱っており，一時点に1レコードずつデータ変換を行なわずに転送する。外部での表現がそのまま内部での表現になる。集合体が1つの全体として取り扱われ，しかも変換が行なわれないので，この形式の入出力は，ストリーム式入出力にくらべて効率がよい。

ストリーム式入出力は通常，取扱いの容易さを得るかわりに，効率を犠牲にしている。各データ項目は別々に転送され，データ変換が必要かどうか調べられる。それに対して，レコード式入出力では，レコード全体が，変換もせずに転送されるため，転送の速度が速められる。

互いに関係のあるいくつかのデータ・セットを含むデータ・バンクに対する入出力操作は，ファイル変数を使用することによって簡単にすることができる。すべての入出力ステートメントは，プログラムの実行中にファイルの値を与えたり，修正したりして，ファイル変数を使用することができる。

コンパイル時の操作

PL／I では，コンパイルの段階での操作が許されており，この時に，原始プログラムのテキスト自体に加えられる操作をプリプロセッサー・ステートメントを用いて指定する。最も簡単で，またおそらくは最もよく使われるプリプロセッサー・ステートメントは，%INCLUDE である（一般に，プリプロセッサー・ステートメントの前には%記号がつけられる）。このステートメントは，%INCLUDE ステートメント自体と置き換えてプログラムの中にテキストを挿入する働きをする。この手法の典型的な使用例として，その客先の標準的な定義のセットから宣言をプログラムの中に複写する場合があげられる。

コンパイル時に行なわれる操作のもう1つの働きとして，プログラム・テキストを取捨選択的にコンパイルすることもできる。たとえば，デバッギング用のステートメントを組み込んだり，削除したりすることを指定することができる。

PL／I の働きのうち簡単ではあるが非常に強力な部分を，コンパイル時の操作で使用することができるので，テキストの生成，または置換えや削除は，これまでのものより精巧で微妙な変形を行なうことができるようになっている。このようにして変形されたものは，各客先によって定義されたPL／I言語の拡張機能と考えることもできる。

割込み操作

最近の計算システムには，例外的な条件が起ったときはいつでもプログラムの実行を中断する機能が備わっている。さらに，これらの機能は，プログラムに例外的な条件を処理させ，中断が起った地点に戻させる働きもする。

PL／I は，様々な例外条件を検出する機能をもっている。プログラマーは，該当する条件が起ったならば割込みを起すよう，条件接頭語を使って指定することができる。ONステートメントを使用すれば，プログラマーは，割込みが起ったときにとるべき処置も指

定することができる。

オペレーティング・システムの機能

PL／I のプログラマーは，システム／360 またはシステム／370 ディスク・オペレーティング・システムがもつ数々の機能を呼び出すことができる。その中でも特に重要なもの，たとえば諸言語間の連絡機能，分類／組合せ機能，およびチェックポイント／再始動機能について，以下に概略を述べる。このコンパイラーに対するプログラマーの手引書の中で，これらの関連機能のすべてが解説されている。

PL／I プログラムは，実行時に COBOL および FORTRAN のルーチンと連絡することもできる（ただし，これらのルーチンは，システム／360 ディスク・オペレーティング・システムに対して IBM が開発したコンパイラーによってコンパイルされたものでなければならない）。PL／I プロシージャーは，COBOL または FORTRAN のルーチンを呼び出すことができるし，また COBOL または FORTRAN の主プログラムまたはルーチンから呼び出されることも可能である。さらに，COBOL データ・セットを作成したり，アクセスしたりするために，PL／I プログラムを使用することもできる。以上の機能はすべて PL／I 言語に備わっているものであり，本書でも取り上げて解説する。また，アセンブラー言語を間に介在させれば，上記以外の言語と PL／I 言語との間での連絡も可能になる。このようなインターフェースについては，プログラマーの手引書の中で解説されている。

オペレーティング・システムが，分類／組合せプログラムを組み込んで生成されている場合は，PL／I プログラマーは分類／組合せ機能を使用することができる。この機能は，PL／I で作成したデータ・セット上のレコード，PL／I プログラムから渡されたデータ，および PL／I プログラムに渡されたデータのいずれに対しても使用することができる。

ある PL／I プログラムを，長時間にわたって実行する場合は，オペレーティング・システムのチェックポイント／再始動機能を利用して，機械またはシステムの故障によって引き起される損失を最少限にとどめることができる。故障のあと処理を再開するチェックポイントを，プログラマーが，プログラム内の任意の位置に選択する。そのチェックポイントから，故障が起きたところまでの間で行なわれた処理だけが，無駄になる。チェックポイントのところまでに得られた結果は，実行を続けるのに必要なデータ（プログラムのコピーおよびそれに関連する記憶域も含めて）といっしょに，補助記憶装置に保存される。

第2章： プログラムの諸要素

PL／I ステートメントは，形式上の制約がほとんどないので，特別なコーディング用紙を使ったり，各々のステートメントが特定の桁から始まっているかどうかチェックしたりする必要なしに，プログラムを書くことができる。各々のステートメントは，セミコロンで止める点以外，形式は全く自由である。前のステートメントの次の桁または位置から，次のステートメントを書き始めてもよいし，間にブランクを何桁入れてもよい。

文字セット

原始プログラムを書く際には，2種類の文字セット（60文字セットと48文字セット）のどちらかを使用することができる。ある外部プロシージャーに，2つのセットのうちどちらを使うかということは任意選択であるが，実際には，この選択は，使用している装置によって決まる。

60文字セット

60文字セットは，数字，特殊文字，および英字から構成されている。

英字として扱われるものには29種類あり，通貨記号（$），ナンバー記号（#），単価記号（@）と，その後に英字の26文字が，拡張2進化10進コード（EBCDIC）形式で，IBMシステム／360における照合順序にしたがってつづいている。英語以外の言語を使用する場合は，$，#，および@を別の文字にかえることもできる。

数字は10個あり，10進数は0から9までの数，2進数は0か1である。

特殊文字は，以下にあげるように，全部で21個ある。

名称	文字
ブランク	
等号または割当て記号	=
正符号	+
負符号	−
星印または乗算記号	*
斜線または除算記号	／
左括弧	(
右括弧	)
カンマ	,
小数点またはピリオド	.
単一引用符 またはアポストロフィ	'
パーセント記号	%
セミコロン	;
コロン	:
"Not"記号	¬
"And"記号	&
"Or" 記号	|
"より大"記号	>
"より小"記号	<
分割文字	_
疑問符	?

特殊文字を組み合せて，別の記号を作ることができる。たとえば，<=は，"より小，または等しい"という意味であり，¬=は，"等しくない"という意味である。**の組合せは，べき乗を表わす（X**2はX^2という意味である）。このような合成記号の中にブランクを入れてはならない。分割文字は，タイプライターのアンダーライン（下線）と同じであり，名前の中などで，読みやすくするためにGROSS_PAYというように使うことができる。

48文字セット

48文字セットは，60文字セットのうちの48文字から構成されている。以下にあげる4つの文字を除いて，数が減らされた文字セットにある文字を組み合わせて，大きな文字セットから抜かされた文字を表現することができる。たとえば，48文字セットにパーセント記号（%）が含まれていないとすれば，斜線を2つ使って（／／）表わすことができる。このように組み合せて表現することのできない文字は，通貨記号，ナンバー記号，分割文字，および疑問符の4つである。

この文字セットに関する制約とその修正方法については，B節「文字セットおよびそのEBCDICとカード穿孔コード」のところで説明する。

名　　称	文字	用　　法
カンマ	,	リスト内の要素を区切る
ピリオド	.	2進数または10進数の小数点を表わす。修飾された名前の要素を連結する。
セミコロン	;	ステートメントの終り
割当て記号	=	値の割当てを表わす。[1]
コロン	:	ステートメントに接頭語を連結する。配列の限界を指定するのに使用することができる。
ブランク		ステートメントの要素を区切る。
単一引用符	'	ストリング定数とピクチャー指定を囲む。
括弧	( )	リストを囲むのに用いられる。様々なキーワードに関連する情報を指定する。演算子およびオペランドと組合わせて，計算式の部分を区切る。
矢印	->	ロケーター修飾を表わす。
パーセント記号	%	コンパイル時にプリプロセッサーによって実行されるステートメントを表わす。

[1]=という文字は，等号としても割当て記号としても使用できることに注意。

図2.1　特殊文字の諸機能

文字セットの用い方

PL／I プログラムを構成する要素はすべて，PL／I 文字セットを使って作り上げられるが，2つだけ例外があり，文字ストリング定数と注釈は，8ビット・コードで表わされる256文字の中のどの文字を含んでいてもよい。

文字の中には，PL／I プログラムの中で特殊な働きをするものがある。たとえば，演算子として働く文字がいくつかある。

演算子には，算術，比較，ビット・ストリング，およびストリングの4種類がある。

算術演算子は，以下の通りである。

+	加算または正の接頭語を表わす
−	減算または負の接頭語を表わす
*	乗算を表わす
／	除算を表わす
**	べき乗を表わす

比較演算子は，以下の通りである。

>	"より大"を表わす
¬>	"より大きくない"を表わす
>=	"より大または等しい"を表わす
=	"等しい"を表わす
¬=	"等しくない"を表わす
<=	"より小または等しい"を表わす
<	"より小"を表わす
¬=	"より小さくない"を表わす

ビット・ストリング演算子は，以下の通りである。

¬	"not" を表わす
&	"and" を表わす
|	"or" を表わす

ストリング演算子は以下の通りである。

||	連結を表わす

図2.1は，上記の他の特殊文字の働きを示したものである。

識別名

PL/I プログラムの中では，データ，ファイル，ステートメント，および別のプログラム域の入口点に，名前またはラベルを与えることができる。名前またはラベルをつける際に，プログラマーは，識別名に関する統語規則を守らなければならない。

識別名は，1桁の英字，または英数字と分割文字からなるストリングで，注釈または定数の中に含まれておらず，前後にブランクまたは何か他の区切り記号があるものである。ストリングの最初の文字は英字でなければならない。また，長さは31桁を越えてはならない。

言語のキーワードもやはり識別名である。キーワードは，該当する文脈の中で使用したときに，コンパイラーに対して特別な意味をもつ識別名である。キーワードは，とるべき処置，データの性質，名前がもつ目的，などを指定するもので，READ，DECIMAL，およびENDFILE などがその例である。略記できるキーワードもある。キーワードとその略称のすべてをリストしたものが，C節「キーワードおよびその略称」に記載されている。

注：PL/Iのキーワードは，予約語ではない。これらは，該当する文脈の中に出てきたときにだけ，キーワードとして認められる。その他の文脈の中では，プログラマーの定義した識別名として使用することができる。

次にあげるのは，名前またはラベルとして使用できる識別名の例である。

```
A
FILE2
LOOP_3
RATE_OF_PAY
#32
```

このコンパイラーでは，後の章で述べるいくつかの識別名については，長さが7桁を越えてはならず，分割文字を含んでいてはならないという制約がある。このような制約を受けるのは，外部名と呼ばれる特定の名前で，これらは，オペレーティング・システムや，別個にコンパイルされた2つ以上のプロシージャーから参照することができるものである。PL/I プロシージャーの外部名が7桁を越えていると，コンパイラーは，先頭の4桁と末尾の3桁を連結して，間の文字を切り捨てる。COBOLまたはFORTRANルーチンの入口名は，最高8桁まで許されている。9桁以上指定すると，左端の8桁だけが用いられる。

ブランクの用い方

ブランクは，PL/I プログラム全体にわたって自由に使用することができる。ブランクは，演算子や他の区切り符号の前後に置くことができる。一般に，1桁でもブランクを置くことができるところであれば（たとえば，1つのステートメント内の各語の間など），いくつブランクを置いてもよい。

識別名や定数が，何か他の区切り符号や注釈によって区切られていないときは，それらを区切るために1つまたは2つ以上のブランクを使用しなければならない。しかし，識別名，定数（文字ストリング定数をのぞく），および合成演算子（¬=など）の中にブランクを入れてはならない。

上記以外に，ブランクが必要な場合，あるいはブランクを入れてよい場合については，その機能を解説する文の中で注記することにする。以下にブランクの用法の例をあげる。

AB+BC と AB + BC は同じである。

TABLE(10)とTABLE (10)は同じである。

FIRST,SECONDとFIRST, SECONDは同じである。

ATOB と A TO B は同じではない。

注釈

注釈は，プログラムの中でブランクを置くことができるところであればどこに置いてもよい（ただし，文字ストリングなどのデータ項目の中はのぞく）。注釈はブランクとして扱われるので，区切り用のブランクが必要なところでかわりに使用することもできる。これ以外に，注釈はプログラムの実行には何の影響も及ぼさない。つまり，注釈は，文書記録上の目的にだけ用いられるものである。

注釈の一般的な形式は，次の通りである。

```
/* 文字ストリング */
```

/*という文字の組合せは，注釈の始まりを示し，その逆の組合せ*/は，注釈の終りを示すものである。どちらの場合も，2つの文字の間にブランクや別の文字を入れて区切ってはならない。つまり，斜線と星印が直接隣り合っていなければならない。注釈自体の中には，*/の組合せ以外の文字ならば何でも入れることができる（*/があると，注釈の終りと解釈されてしまう）。次にあげるのは，注釈の例である。

```
/* THIS WHOLE SENTENCE COULD
   BE INSERTED AS A COMMENT */
```

使用する機械構成で許される文字は何でも，注釈の中に書くことができる。

注：注釈の始まりを示す／＊という文字の組合せは，入力レコードの1～2桁目に入れると，入力の流れの中にあるデータを区切るために用いられる／＊ジョブ制御ステートメントと取り違えられる恐れがあるので，1～2桁目に入れてはならない。

プログラムの基本的な構造

PL／I のプログラムは，ステートメントと呼ばれる基本的なプログラムの要素から作り上げられている。ステートメントには，単純ステートメントと複合ステートメントの2つのタイプがある。これらのステートメントが集まって，グループおよびブロックと呼ばれるさらに大きなプログラム要素を作り上げる。

単純ステートメントと複合ステートメント

単純ステートメントには，キーワード，割当て，およびナル，の3種類のステートメントがあり，いずれの場合も，セミコロンで終るステートメント本体をもっている。

キーワード・ステートメントは，そのステートメントの機能を表わすキーワードをもつもので，残りの部分がステートメント本体になる。

割当てステートメントは，割当て記号（＝）がその中に含まれているもので，キーワードをもっていない。

ナル・ステートメントは，セミコロンだけからなるステートメントで，操作を何も指示しない。セミコロンがステートメント本体である。

次に示すのは，単純ステートメントの例である。

GO TO LOOP_3;　（GO TOはキーワードである。GOとTOの間のブランクは任意選択である。ステートメント本体は，LOOP_3；である。）

A = B + C;　（割当てステートメント）

合成ステートメントは，そのステートメント本体の一部として1つまたは2つ以上の他のステートメントが含まれるものである。合成ステートメントには，IFステートメントとONステートメントの2つがある。合成ステートメントの最後のステートメントは，セミコロンで終る単純ステートメントである。したがって，合成ステートメントは，このセミコロンで終了するわけである。以下の例に示すように，IFステートメントには2つのステートメント（単純でも合成でもよい）を含めることができる。

```
IF A > B THEN A = B + C;
        ELSE GO TO LOOP_3;
```

次にあげるのは，ONステートメントの例である。

```
ON OVERFLOW GO TO OVFIX;
```

ステートメントの接頭語

単純，合成，いずれのステートメントの場合も，1つまたは2つ以上の接頭語をつけることができる。接頭語には，ラベル接頭語と条件接頭語の2種類がある。

ラベル接頭語は，プログラム内のどこか他の地点からステートメントを参照することができるように，ステートメントを識別する識別名で，該当するステートメントの前に書き，コロンをつけてそのステートメントに連結する。どのステートメントにも，1つまたは2つ以上のラベルをつけることができる。2つ以上指定した場合は，それらのどれを使っても，そのステートメントを参照することができる。

条件接頭語は，そこで指名されている条件が起ったときに割込みを起すかどうかを指定するものである。条件名は言語のキーワードであり，それぞれ，プログラムの実行中に起り得る例外的な条件を表わしている。OVERFLOW，SIZE などがその例である。OVERFLOW条件は，浮動小数点数の指数が，許される最大値（約 10^{75} ）を越えたときに起こる。SIZE条件は，ある値を変数に割り当てたときに，高位の数字やビットが失なわれてしまった場合に起こる。

ある条件が起らないと確信できる場合は，プログラマーは，その条件接頭語の名前の前にNOという語を書いて，その条件を禁止することができる。NOを使用するときに，NOと条件名の間にブランクを入れてはならない。

条件接頭語は，1つまたは2つ以上の条件名をカンマで区切り，括弧で囲んだリストから構成される。1つのステートメントに1つまたは2つ以上の条件接頭語をつけることができるが，括弧で囲んだリストの後にはそれぞれコロンをつけなければならない。条件接頭語は，ステートメント全体の前（ラベル接頭語がいくつかついているときは，そのすべてより前）に書かなければならない。次にあげるのは，その一例である。

```
(SIZE,NOOVERFLOW): COMPUTE:
                       A=B*C**D;
```

この条件接頭語（この例では1個）は，この割当てステートメントの実行中にSIZE条件が生じたときは割込みを起すが，OVERFLOW条件が生じたときは割込みを起さないことを指示している。ラベル接頭語 COMPUTE より前に条件接頭語があることに注意していただきたい。

接頭語とそれに結びつけられるステートメントの間にあるブランクは無視されるので，カード入力を使用するときは，そのステートメントを穿孔するカードの前に別のカードを入れ，そこに条件接頭語を穿孔すると便利である場合が多い。このようにすれば，デバッギングの後で，接頭語を容易に取り除くことができる。たとえば，

```
(NOCONVERSION):
(SIZE,NOOVERFLOW):
COMPUTE:    A=B*C**D;
```

ここでは，条件接頭語が2つあることに注意していただきたい。最初の接頭語は，データ変換の試行中に無効な文字があっても割込みを起さないよう指定している。

個々の条件接頭語については，第14章「例外条件の処理とプログラム・チェックアウト」のところで説明する。

グループとブロック

グループは，DOステートメントで始まり，該当するENDステートメントで終る一連のステートメントであり，制御の目的に使用される。DOグループと呼ばれることもある。

ブロックは，プログラムの領域を定義する一連のステートメントであり，名前の有効範囲を区切るため，および制御の目的に使用される。1つまたは2つ以上のブロックが集って，プログラムを構成する。どのようなステートメントも，ブロックの中に入っていなければならない。ブロックには，開始ブロックとプロシージャー・ブロックの2種類がある。開始ブロックは，BEGINステートメントとENDステートメントによって区切られる。プロシージャー・ブロックは，PROCEDUREステートメントとENDステートメントによって区切られる。開始ブロックは，どのような場合も，プロシージャー・ブロックの中に含まれていなければならない。

開始ブロックの場合は，前から順に実行されて開始ブロックの中に入り，そのまま再び外へ出てゆく。しかし，プロシージャー・ブロックの場合は，別のブロック内のステートメントを実行することによって，呼び出さなければならない。プログラムで実行される最初のプロシージャーは，オペレーティング・システムによって自動的に呼び出される。この最初のプロシージャーは，PROCEDUREステートメントの中でOPTIONS（MAIN）を指定することによって識別しなければならない。

第3章： データの諸要素

データは，一般に，情報または値を表現するものとして定義される。

PL／I では，データ項目（算術またはストリング）は，変数または定数を用いて参照される。（この場合の変数，定数ということばは，普通の数学でいうことばとは少し違っている。）

変数は，プログラムの実行中に変更することができる値をもつ記号名である。

定数（記号名のこともある）は，変更できない値をもつものである。

次のステートメントには，変数と定数の両方が含まれている。

```
AREA = RADIUS**2*3.1416;
```

AREAおよびRADIUSは変数である。2および3.1416という数は定数である。RADIUS の値はデータ項目であり，計算結果はAREA の値として割り当てられるデータ項目になる。ステートメントの中にある3.1416という数は，それ自身が1つのデータ項目である。

3.1416という数を，プログラムの中で2回以上使用する場合は，3.1416という値が割り当てられた変数として表現する方が都合がよいこともある。したがって，上記のステートメントは，次のように書きかえてもよい。

```
PI = 3.1416;
AREA = RADIUS**2*PI;
```

最後のステートメントの中では，2という数だけが定数である。

定数は，単に値を記述するだけでなく，そのデータ項目の様々な性質も表示する。たとえば，3.1416は，データのタイプが算術タイプであること，このデータ項目は5桁の10進数で，その中の4桁が小数点以下であることを示している。

記号名で表わされる定数は，コンパイラーによって決められた値をもっているが，プログラマーはその内容を知らなくてもよい。このような定数は，通常，プログラムの制御に関するものであり，計算に使う値ではなく内部記憶域のアドレスを表わしている。たとえば，次にあげる例の中にある"LOOP"という識別名は，"A=2*B"というステートメントから生成される機械コードのアドレスである定数を表わす記号名である。

```
       GET LIST (B);
LOOP:  A=2*B;
       C=B+6;
```

変数または記号定数の性質は，その名前の中で直接明らかにはされていない。これらの性質（属性と呼ばれる）をコンパイラーに知らせなければならないので，DECLARE ステートメントの中で特定のキーワードや式を用いて属性を指定することができる。個々のデータ・タイプを記述するための属性について，この章では簡単に説明する。個々の属性についての詳細は，I節「属性」のところをみていただきたい。

PL／I でプログラムを書くにあたって，プログラマーは，許されているデータのタイプ，データの編成方法，およびデータの参照方法などについてよく知っておかなければならない。次に，これらの特徴について述べる。

データのタイプ

PL／I のプログラムで使用されるデータのタイプは，問題データとプログラム制御データの2つのカテゴリーにわけられる。問題データは，プログラムで処理される値を表わすために使用されるものであり，算術データとストリング・データの2つのデータ・タイプがある。プログラム制御データは，プログラム実行の制御をするためにプログラマーが使用するものである。プログラム制御データのタイプには，ラベル，事象，ファイル，入口，ロケーター，エリアの6つがある。

問題データ

問題データのタイプには，算術データとストリング・データの2つがある。

算術データ

算術データ項目は，その値が数値である項目である。算術データ項目は，基数，スケール，精度およびモードに関する性質をもっている。算術変数によって表わされるデータ項目の性質は，その名前に対して宣言された属性によって指定されるか，省略時解釈が適用される。

算術データ項目の基数は，10進数または2進数のどちらかである。

算術データ項目のスケールは，固定小数または浮動小数のどちらかである。固定小数点データ項目は，小数点の位置が指定されている（定数の場合はその中で指定されており，変数の場合はスケール因数によって宣言される）数である。浮動小数点データ項目は，指数（符号は任意選択）を後につけた数である。指数は，仮想上の小数点の位置を，現在の位置から相対的に指定するものである。

算術データ項目の精度は，固定小数の場合にはそのデータ項目が含むことのできる桁数であり，浮動小数の場合には最小限保持しておくべき有効数字の桁数（指数部はのぞく）である。固定小数点データ項目の場合は，精度を用いて，その数の右端の桁から相対的に数えた仮想上の小数点の位置を指定することもできる。

データ項目が固定小数点の変数に割り当てられるときはつねに，宣言された精度が維持される。割り当てられる項目は，小数点の位置に揃えられる。割り当てられる項目の整数部分が，宣言されたものより少ないときは，先行ゼロが挿入される。小数部分が少ないときは，末尾にゼロが挿入される。割り当てられる項目の整数が多すぎるときは，SIZE エラーが起こる可能性がある。小数の桁数が多すぎるときは，右端で切捨て（四捨五入なし）られる。

算術データ項目のモードは，実数または複素数のどちらかである。実数データ項目は，実数の値を表わす数である。複素数データ項目は，2つの数（一方は実数で，もう一方は虚数）を組み合わせたものである。複素数データ項目を表わす変数の場合は，2つの部分の基数，スケール，および精度が同じでなければならない。

算術変数の基数，スケール，およびモードは，キーワードによって指定する。精度は，括弧に入れた10進整定数を用いて指定する。算術変数および定数の精度について，以下に詳しく説明する。

このあと説明する実数の算術データ・タイプは，10進固定小数，2進固定小数，10進浮動小数，および2進浮動小数である。これらはいずれも，複素数データ項目の実数部分としても使用することができる。複素数の虚数部分については，この章の「複素数の算術データ」のところで説明する。

複素数の算術変数は，COMPLEX 属性をつけて明示的に宣言しなければならない。実数の算術変数は，REAL 属性をもつことを明示的に宣言することができるが，とくに複素数であると明示宣言されていない限り，算術変数は一般に実数とみなされるので，通常は，わざわざ宣言する必要はない。

10進固定小数点データ

10進固定小数点データは，1桁または2桁以上の10進数（小数点は任意選択）からなっている。小数点がないときは，右端の桁のすぐ右側に小数点があるものとみなされる。10進固定小数点定数の前に，符号をつけてもよい。

次にあげる例は，プログラムの中に書かれる10進固定小数点定数である。

3.1416

455.3

732

003

−5280

.0012

式を評価するとき，10進固定小数点定数は，精度（p，q）が明らかにされる（pは定数の中にある桁数の合計で，qは小数点の右側に指定されている桁数である）。たとえば，

3.14 の精度は，（3，2）である。

10進固定小数点変数を宣言するためのキーワード属性は，DECIMALとFIXEDである。精度は，括弧の中に2つの10進定数を書き，カンマで区切って記述する。最初の数は総桁数を指定するもので，符号をつけてはならない。2番目の数はスケール因数で，符号をつけてもよく，小数点の右側にある桁数を指定する。変数が整数を表わす場合は，スケール因数とその前にあるカンマを省略することができる。諸属性はどのような順序で書いてもよいが，精度の指定は，DECIMALかFIXED（またはREALかCOMPLEX）のすぐ後に書かなければならない。

プログラムの中で書かれる10進固定小数点変数の宣言の例は，次の通りである。

```
DECLARE A FIXED DECIMAL (5,4);

DECLARE B FIXED (6,0) DECIMAL;

DECLARE C FIXED (7,-2) DECIMAL;

DECLARE D DECIMAL FIXED REAL (3,2);
```

最初のDECLARE ステートメントは，Aという識別名が，5桁以下の10進固定小数点項目で，そのうちの4桁は小数部であること（つまり仮想上の小数点の右側にあること）を表わしている。Aに割り当てられる項目はいずれも10進固定小数に変換され，小数点の位置に揃えられる。

2番目のDECLARE ステートメントは，Bが6桁以下の整数であることを表わしている。カンマとゼロは不要であることに注意していただきたい。これは，B FIXED DECIMAL (6)と指定してもよい。

3番目のDECLARE ステートメントは，負のスケール因数－2を指定している。これは，仮想上の小数点の位置が，この項目の右端の桁のさらに右2桁目のところにあることを意味している。

4番目のDECLARE ステートメントは，Dが，3桁以下の固定小数項目で，そのうちの2桁が小数であることを表わしている。

10進数の最大許容桁数は，15桁である。指定がない場合にとられる省略時解釈の精度は，(5，0)である。10進固定小数算術データの内部コード形式は，パック10進数である。パック10進数は，1バイトについて2桁ずつ記憶され，右端のバイトの右4ビットに符号が表示される。したがって，10進固定小数データ項目は，その変数が宣言されたときに，(p)で偶数の桁数が指定されていても，つねに奇数の桁数をもつものとして貯えられる。

偶数桁が宣言されている場合は，最高位の桁に余分の桁がもうけられることになり，このデータ項目に対して行なわれる操作（比較操作など）のすべてにこの桁も参加する。このような余分の高位桁に算術的な桁あふれが及んだり，割り当てが行なわれたりしたことは，SIZE条件が設定されているときにだけ，検出することができる。

2進固定小数点データ

2進固定小数点定数は，1桁または2桁以上の2進数（符号は任意）と，そのすぐ後に間をあけずにBという文字をつけたものから構成される。符号は，必要ならば，定数の前につけることができる。

次にあげるのは，プログラムの中で書かれる2進固定小数点定数の例である。

```
10110B

11111B

101B

-111.01B

1011.111B
```

式を評価する際，2進固定小数点定数は，精度（p，q）が明らかにされる（pはその定数の中にある桁数の合計，qは小数点の右側にある2進数の桁数）。たとえば，

0000001Bは，(7，0）の精度をもっている。

2進固定小数点変数を宣言するためのキーワード属性は，BINARYとFIXEDである。精度は，2進数の最大桁数と小数点の右側の桁数を表わす2つの10進整定数を括弧に入れて書いて指定する。変数が整数を表わしているときは，2番目の数とカンマを省略してよい。諸属性はどのような順で指定してもよいが，精度の指定は，BINARYかFIXED（またはREALかCOMPLEX）の後に記入しなければならない。次にあげるのは，2進固定小数点変数の宣言の例である。

```
DECLARE FACTOR BINARY FIXED
                         (20,2);
```

FACTOR は，最高20桁（そのうちの2桁が小数以下）の2進数の算術データ項目を表わすことのできる変数であると宣言されている。この値の範囲を10進数で表わすと，－262，144.00から＋262，143.75である。

2進数の最大許容桁数は，31である。省略時解釈の精度は，(15，0)である。2進固定小数点データの内部コード化算術形式は，ハーフワードまたはフルワードの2進固定小数である。ハーフワードは15ビットと符号ビット，フルワードは31ビットと符号ビットである。精度が(15，0)以下の2進固定小数点データ項目は，ハーフワードとして貯えられ，精度が(15，0)を越えた（最大の精度の範囲内で）ものは，フルーワードとして貯えられる。宣言された桁数は，低位の位置にあるものとみなされるが，高位の余分の桁も，このデータ項目に対して行なわれるすべての操作に参加する。このような余分の桁への算術的な桁あふれは，SIZE条件が設定されている場合にだけ検出することができる。

標準の省略時解釈のルールが適用される場合は，宣言が行なわれていない識別名は，最初の文字がＩからＮまでのいずれかであれば，省略時解釈の精度をもつ２進固定小数点変数とみなされる。

１０進浮動小数点データ

１０進浮動小数点定数は，１０進数のフィールドの後にＥという文字をつけ，さらにその後に１０進整数の指数（任意選択により符号をつける）をつけた形で書かれる。最初のフィールドには，小数点を入れてもよく，また，定数全体の前に正または負の符号をつけてもよい。次にあげるのは，プログラムの中に書かれる１０進浮動小数点定数の例である。

１５Ｅ－２３

１５Ｅ２３

４Ｅ－３

－４８３３３Ｅ６５

４３８Ｅ０

３１４１５９３Ｅ－６

.００３１４１５９３Ｅ３

最後の２つの例は，同じ値を表わしている。

式が評価されるとき，１０進浮動小数点定数の精度（p）が明らかになる（ｐは，Ｅという文字の左側にある定数の桁数（小数）である）。たとえば，

０.０１２Ｅ５は，(4)という精度をもっている。

１０進浮動小数点変数を宣言するためのキーワード属性は，ＤＥＣＩＭＡＬとＦＬＯＡＴである。精度は，１０進整定数を括弧で囲んだものを用いて記述する。これは，維持すべき有効数字の最小桁数を指定するものである。変数に割り当てられる項目のフィールドが，その変数について宣言された精度よりも大きい場合は，右側で切捨てが行なわれる。つまり，最下位の有効数字が最初に切り捨てられる。諸属性はどのような順序で記入してもよいが，精度の指定は，ＤＥＣＩＭＡＬかＦＬＯＡＴ（またはＲＥＡＬかＣＯＭＰＬＥＸ）の後に書かなければならない。次にあげるのは，１０進浮動小数点変数の宣言の例である。

```
DECLARE LIGHT_YEARS DECIMAL
                        FLOAT(5);
```

このステートメントは，ＬＩＧＨＴ_ＹＥＡＲＳが，最低５桁の有効数字のくわしさをもつ１０進浮動小数点データ項目を表わしていることを指定している。

１０進浮動小数点データ項目に許される最大の精度は，（１６）であり，省略時解釈の精度は，（６）である。指数は，２桁を越えてはならない。１０進浮動小数点データ項目によって表わすことのできる値の範囲は，およそ10^{-78}から10^{75}である。１０進浮動小数点データの内部コード化算術形式は，正規化された16 進数の浮動小数点であり，最初の１６進数桁の左側に小数点があるものとみなされる。宣言された精度が（6)以下の場合は，短精度の浮動小数点形式が用いられる。宣言された精度が（6)より大きい場合は，長精度の浮動小数点形式が用いられる。

標準の省略時解釈のルールが適用されている場合は，宣言されていない識別名のうち，先頭の文字がＡからＨ，ＯからＺ，または英字の拡張文字＄，＃，＠のいずれかであるものは，１０進浮動小数点変数とみなされる。

２進浮動小数点データ

２進浮動小数点定数は，２進数のフィールドの後にＥという文字をつけ，さらにその後に１０進整数の指数（符号は任意）とＢという文字をつけた形をしている。指数は１０進整数で，２のべきを表わしている。２進数のフィールドには，小数点を含めることもできる。定数全体の前に正または負の符号をつけることもできる。次にあげるのは，プログラムの中で書かれる２進浮動小数点定数の例である。

１０１１０１Ｅ５Ｂ

１０１.１０１Ｅ２Ｂ

１１１０１Ｅ－２８Ｂ

－１０.０１Ｅ９９Ｂ

式を評価するとき，２進浮動小数点定数の精度（p）が明らかにされる（ｐは，Ｅの左側にある２進数の桁数（小数部）である）。たとえば，

０.０１０１Ｅ３３Ｂは，精度（５)をもっている。

２進浮動小数点変数を宣言するためのキーワード属性は，ＢＩＮＡＲＹとＦＬＯＡＴである。精度は，維持すべき有効数字の最低桁数を指定する１０進整定数を括弧で囲んだもので表わされる。

諸属性はどのような順で記入してもよいが，精度の指定は，BINARYかFLOAT（またはREALかCOMPLEX）のすぐ後に書かなければならない。次に示すのは，2進浮動小数点変数の宣言の例である。

```
DECLARE S BINARY FLOAT (16);
```

上の例は，識別名Sが，2進数フィールドの中で16桁の2進浮動小数点データ項目を表わすことを指定している。

2進浮動小数点データ項目で許される最大の精度は（53）で，省略時の精度は（21）である。指数部は，10進数で3桁を越えてはならない。2進浮動小数点データ項目では，ほぼ2^{-260}から2^{252}までの範囲の値を表わすことができる。2進浮動小数点データの内部コード化算術形式は，正規化された16進数の浮動小数点数である。宣言された精度が（21）以下の場合は，短精度の浮動小数点形式が用いられ，宣言された精度が（21）を越えている場合は，長精度の浮動小数点形式が用いられる。

複素数の算術データ

複素数モードでは，算術データ項目は，実数部と符号付の虚数部の2つの部分からなると考えられる。PL／Iでは，複素数の定数はない。複素数の値は，実数の定数と虚数の定数を組み合わせて得られる。

虚数の定数は，実数の定数（どのようなタイプでもよい）のすぐ後にIという文字を書いて示す。

次に示すのは，プログラムの中に書かれる虚数の定数の例である。

```
27I

3.968E10I

11011.01BI
```

これらはそれぞれ，ゼロの実数部をもつと考えられる。ゼロ以外の実数部をもつ複素数の値は，次のような形式で表わされる。

〔＋｜－〕 実数の定数 ｛＋｜－｝ 虚数の定数

したがって，38+27I というように複素数の値を書くことができる。

複素数の変数を宣言するためのキーワード属性は，COMPLEXである。様々なタイプの実数算術データで有効な属性はいずれも，複素数の変数でももつことが許される。基数，スケール，および精度の属性はそれぞれ，どちらのフィールドにも適用される。

COMPLEX 属性をもつと明示的に宣言されていない限り，変数は，実数データ項目を表わすとみなされる。

数字データ

数字データ項目（数値フィールド・データ項目ともいう）は，PICTURE 属性をもち，数値ピクチャーが指定されて宣言されている変数の値である。このデータ項目は，10進数の固定小数または浮動小数の値を文字で表現したものである。

数値ピクチャー指定は，算術値をもつ（または算術値に変換できる）データだけがそこに割り当てられるような文字ストリングを記述するものである。ピクチャー文字AまたはXは，数値以外のピクチャー文字ストリングに対して使用されるものなので，数値ピクチャー指定には，これらの文字を含めることはできない。数値ピクチャー指定の基本形式は，数字を指定するピクチャー文字9を1つまたは2つ以上つづけたものに，仮想上の小数点の位置を表わすピクチャー文字Vをつけて（Vは任意選択）表わされる。ピクチャー指定は，単一引用符で囲まなければならない。たとえば，

```
'999V99'
```

この数値ピクチャー指定は，右から2桁目の位置に小数点があるとみなされる，最高5桁の文字形式の10進数からなるデータ項目を記述するものである。

数値ピクチャー指定の中で，繰返し因数を使用することができる。繰返し因数は，その後に書かれたピクチャー文字をいくつ繰り返し使用するかを示す10進数の整定数であり，括弧で囲んで指定する。たとえば，次にあげるピクチャー指定は，上記の例の記述と同じ結果になる。

```
'(3)9V(2)9'
```

数字変数を宣言するための形式は，次の通りである。

```
DECLARE 識別名 PICTURE
  '数値ピクチャー指定';
```

たとえば，

```
DECLARE PRICE PICTURE '999V99';
```

上の例は，PRICEに割り当てられる値はいずれも，右から2桁

目のところに仮想上の小数点がある5桁の10進数からなる文字ストリングとされることを指定している。PRICEに割り当てられるデータは，固定小数の10進数データの場合と同じようにして，仮想上の小数点の位置に合わせられる。

数値ピクチャー指定は，定数を書くときとほぼ同じ形式で，データの算術属性を指定する。ピクチャー文字によって表わすことができるのは，10進数データだけである。複素数データは，固定小数点または浮動小数点データ項目のどちらかを記述する単一のピクチャー指定といっしょにCOMPLEX 属性を指定することによって，宣言することができる。数字データ項目の中で使用できる10進数の最大桁数は，15桁である。

数字データは算術属性をもってはいるが，コード化算術形式で貯えられるのではないことに注意していただきたい。数字データはゾーン10進数形式で貯えられるので，算術演算に使用するためにはその前に，パック10進数か16進数の浮動小数点形式のどちらかに変換しなければならない。このような変換は自動的に行なわれるが，余分な実行時間を必要とする。

数字データは，文字ストリングと同じように文字形式になっているし，また，コード化された算術データと同じように小数点の位置に合わせられるが，コード化された算術データ項目とも，文字ストリングとも別な形で処理される。数字データ項目に挿入するために編集文字を指定することができるが，この文字は，実際にデータ項目の中に貯えられる。したがって，項目を印刷したり，文字ストリングとして処理したりするときは，その割当ての中に編集文字も含められる。しかしながら，数字項目が，別の数字変数または算術変数に割り当てられるときは，その割当ての中に編集文字は含まれず，実際の数字と仮想上の小数点の位置だけが割り当てられる。次の例について考えてみよう。

```
DECLARE PRICE PICTURE '$99V.99',
        COST CHARACTER (6),
        VALUE FIXED DECIMAL (6,
        2);

PRICE = 12.28;

COST = '$12.28';
```

PRICEに対するこのピクチャー指定の中では，通貨記号($)と小数点(・)が編集文字である。これらは，データ項目の中に文字として貯えられる。しかし，これらは算術値の一部ではない。2番目の割当てステートメントを実行したあと，PRICEおよびCOSTの実際の内部文字表現は，全く同じになると考えることができる。この2つを印刷すれば，全く同じに印刷される。しかし，この2つがつねに同じ働きをするとは限らない。たとえば，

```
VALUE = PRICE;

COST  = PRICE;

VALUE = COST;

PRICE = COST;
```

最初の2つの割当てステートメントが実行されたあとでは，VALUEの値は0012.28になり，COSTの値は'$12.28'になる。PRICEをVALUEに割り当てるときは，通貨記号と小数点は編集文字とみなされ，これらは割当ての一部にはならず，PRICEの算術値が内部コード化算術形式に変換される。しかしながら，PRICEからCOSTに割り当てるときは，この割当ては文字ストリングに対するものであるから，数値ピクチャー指定の編集文字はつねにこのような割当てに参加する。PRICEは文字形式で貯えられるので，変換は不要である。

3番目と4番目の割当てステートメントは，エラーを引き起す。ストリングの中に通貨記号があると，算術定数としては無効になるので，COSTの値をVALUEに割り当てることはできない。全く同じ理由から，COSTの値をPRICEに割り当てることはできない。数値ピクチャー指定を伴って宣言された変数に割り当てることができるのは，算術タイプの値または算術タイプに変換できる値だけである。

注：小数点は，編集文字であっても，文字ストリングの中の実際の文字であってもよいが，算術定数の中に小数点があることが許されているので，算術形式に変換する際に，このためにエラーが起こることはない。正または負の符号についても，算術定数の前に符号があってよいので，同じことがいえる。

数値ピクチャー指定の中では，ゼロ消去文字，浮動文字，挿入文字などの他の編集文字を使用することができる。ピクチャー文字に関する詳細は，D節「ピクチャー指定文字」およびI節「属性」の中のPICTURE 属性の項を読んでいただきたい。

ストリング・データ

ストリングは，1つのデータ項目として取り扱うことのできる，連

続する一連の文字（または2進数）である。ストリングの長さは，その中に含まれる文字（または2進数）の数である。

ストリングには，文字ストリングとビット・ストリングの2種類がある。

文字ストリング・データ

文字ストリングには，特定の機械構成で文字として認められるものであれば，どのような数字，英字，または特殊文字でも含めることができる。文字ストリングの中にあるブランクはいずれも必須の文字と考えられ，長さのカウントに含められる。文字ストリングの内部に注釈を挿入しても，それは注釈とは認められない。注釈は，注釈の区切り（/*と*/）も含めて，文字ストリング・データの一部分とみなされてしまう。

文字ストリング定数をプログラムの中に書くときは，単一引用符で囲まなければならない。ストリングの中に文字として単一引用符を書きたいときは，間にブランクをあけずに単一引用符を2つ書かなければならない。文字ストリングの長さは，両端の引用符の中にある文字の数である。ストリングの中で2つの単一引用符を使って1個の単一引用符を表わしている場合は，2つで1個の文字として数えられる。

次にあげるのは，文字ストリング定数の例である。

```
'LOGARITHM TABLE'

'PAGE 5'

'SHAKESPEARE''S ''''HAMLET'''''

'AC438-19'

(2)'WALLA '
```

3番目の例は，実際にはSHAKESPEARE'S ''HAMLET''（長さは24）を表わしている。最後の例の中で括弧の中に入っている数は，その後に書かれている文字を何度繰り返すかを示す繰返し因数である。この例では，'WALLA WALLA '（ブランクも繰り返される文字の1つである）という定数が指定されている。繰返し因数は，括弧で囲んだ符号なしの10進整定数でなければならず，許される最大の値は，32,767である。ナル文字ストリングは，間にブランクをあけずに2つの引用符を書いて示す。

文字ストリング変数を宣言するためのキーワード属性は，CHARACTERである。ストリングの中にある文字の数を指定する式または10進整数を括弧に入れて書くことによって，長さを宣言することもできる。長さの指定は，CHARACTERというキーワードの後につづけて書かなければならない。たとえば，

```
DECLARE NAME CHARACTER (15);
```

このDECLARE ステートメントは，NAMEという識別名が，長さ15桁の文字ストリング・データ項目を表わすことを指定している。15桁より短い文字ストリングがNAMEに割り当てられるときは，左につめられ，長さが15桁になるまで右側にブランクがパッディングされる。15桁より長いストリングが割り当てられるときは，右側がその分だけ切り捨てられる。（このような切捨てが行なわれたときは，STRINGSIZE条件を使用していれば検知することができる。）

長さの指定がないときは，長さは1とみなされる。

文字ストリング変数は，次のようにして，VARYING 属性をもつことを宣言することもできる。

```
DECLARE NAME CHARACTER (15)
                    VARYING;
```

このDECLARE ステートメントは，最大の長さが15で長さの変わる文字ストリング・データ項目を表わすために，NAMEという識別名を使用することを指定している。ある特定の時点におけるNAME の実際の長さ属性は，その時点で割り当てられているデータ項目の長さになる。プログラマーは，長さの変わる文字ストリングの長さを記録しておく必要はない。これは，自動的に行なわれる。しかし，プログラマーは，LENGTH組込み関数を用いて，ある任意の時点における長さを決めることができる。（この関数については，第13章「編集およびストリングの取扱い」のところで解説する。）長さの変わるストリング変数は，初期値が与えられるまで，その長さが定義されていないことに注意していただきたい。

文字ストリング・データは，内部的には文字形式，すなわち各々の文字が記憶装置の中でそれぞれ1バイトを占める形で貯えられる。CHARACTER属性をつけて宣言された変数に許される最大の長さは，32,767である。繰返し因数が適用される前の文字ストリング定数に許される最大の長さは，そのコンパイラーが使用できる記憶域の量によって異なるが，最低の場合でも512である。文字ストリングの最小の長さはゼロである。長さの変わるストリングに割り振られる記憶域は，宣言された最大の長さよりも2バイトだけ多くなる。最初の2バイトには，そのストリングの現在のバイト数が入っている。

文字ストリング変数は，次の形式のPICTURE属性を用いて宣言することもできる。

PICTURE '文字ピクチャー指定'

文字ピクチャー指定は，A，X，および9というピクチャー指定文字からなるストリングである。ピクチャー文字のストリングは，単一引用符で囲んでおかなければならない。また，その中には最低1個のAまたはXが入っていなければならないが，その他に9以外の文字が入っていてはならない。Aという文字は，記述されているフィールドの中の該当位置に，英字またはブランクが入ることを指定するものである。Xという文字は，そのフィールドの該当位置にどのような文字があってもよいことを指定するものである。9というピクチャー文字は，該当位置に数字またはブランクが入ることを指定している。たとえば，

```
DECLARE PART_NO PICTURE
                    'AA9999X999';
```

このDECLAREステートメントは，PART_NOという識別名が，2桁の英字，4桁の数字，1桁の文字（どんな文字でもよい），および3桁の数字からなる文字ストリング・データ項目を表わすことを指定している。

ピクチャー指定の中での繰返し因数の用い方は，ストリング定数の中での用い方とは異なっている。繰返し因数は，引用符の内側に置かなければならない。この場合の繰返し因数は，すぐ後にあるピクチャー文字を繰り返すことを指定する。たとえば，上記のピクチャー指定は，次のように書くこともできる。

```
'(2)A(4)9X(3)9'
```

ピクチャー指定で許される最大の長さは，前に述べたように，文字ストリング定数で許される長さと同じである。

文字ピクチャー指定では，ピクチャー文字9は数字またはブランクを指定するものであるが，数値ピクチャー指定の場合は，この文字は数字だけを指定する。

ビット・ストリング・データ

ビット・ストリング定数は，単一引用符で囲んだ一連の2進数のすぐ後にBという文字をつけた形で，プログラムの中に書かれる。

ナル・ビット・ストリング定数は，間をあけずに2個の引用符を書き，そのすぐ後にBという文字をつけた形で書かれる。

次にあげるのは，プログラムの中に書かれたビット・ストリング定数の例である。

```
'1'B

'11111010110001'B

(64)'0'B

''B
```

3番目の例の中で，括弧に入れられた数は，その後にある一連の数字を指定された因数だけ繰り返すことを指定する繰返し因数である。上の例では，64個の2進数ゼロからなるストリングができる。

ビット・ストリング変数は，BITキーワード属性をつけて宣言される。そのストリングの中にある2進数の桁数を指定するために，式または10進整定数を括弧で囲んだものを用いて，長さを宣言することができる。Bという文字は，ストリングの一部ではないので，長さの指定には含まれない。長さの指定は，BITというキーワードの後に書かなければならない。次にあげるのは，ビット・ストリング変数の例である。

```
DECLARE SYMPTOMS BIT (64);
```

文字ストリングと同様，ビット・ストリングは，左から右へ順番に変数に割り当てられる。その変数に対して宣言された長さよりもストリングの方が長い場合には，右端の文字がその分だけ切り捨てられる。短い場合は，右側にゼロがパッディングされる。

長さが指定されていないときは，1とみなされる。

ビット・ストリング変数にVARYING属性を与えて，それが，長さの変わるビット・ストリングを表わすために使用されることを表示することができる。その用い方は，前の項で文字ストリング変数について述べたものと同じである。

ビット・ストリングは，1バイトについて8ビットずつ貯えられる。ビット・ストリング変数に許される最大の長さは，32,767である。繰返し因数が適用される前のビット・ストリング定数に許される最大の長さは，そのコンパイラーで使用できる記憶域の量によって異なるが，最低の場合でも4096（512バイト）である。ビット・ストリングに対する最小の長さは，ゼロである。長さの変わるストリングに対する記憶域は，宣言された最大の長さよりも2バイトだけ大きくなる。最初の2バイトには，そのストリングの現在の長さが，ビット数を単位として入れておかれる。

プログラム制御データ

プログラム制御データには，ファイル，ラベル，入口，事象，ロケーター，およびエリアの各タイプがある。

ファイル・データ

ファイル・データ項目は，PL／I ファイルに関する情報を表わすものであり，ファイル定数であることも，ファイル変数であることもある。ファイル定数は，ファイル変数に割り当てることができる。ファイル変数を参照することは，割り当てられたファイル定数を参照することになる。

ラベル・データ

ラベル・データ項目は，ラベル定数またはラベル変数の値である。

ラベル定数は，ステートメントの接頭語として書かれた識別名であり，実行中にそのラベルを参照することによってそのステートメントにプログラムの制御を渡すことができるようになっている。コロンを使ってラベルを該当のステートメントに結びつける。

```
ABCDE:  DISTANCE = RATE*TIME;
```

この例の中で，ABCDEはステートメント・ラベルである。このステートメントは，命令を正規の順序で実行してくるか，または GO TO ステートメントを使ってプログラム内のどこか別の地点からこのステートメントに制御を渡すことによって，実行することができる。

上記のように使用されているときは，ABCDEはさらにステートメント・ラベル定数として区別することができる。ステートメント・ラベル変数は，ステートメント・ラベル定数を参照する識別名である。次の例について考えてみよう。

```
LBL_A:    ステートメント;
               .
               .
               .
LBL_B:    ステートメント;
               .
               .
               .
          LBL_X = LBL_A;
               .
               .
               .
          GO TO LBL_X;
               .
               .
               .
```

LBL_AとLBL_Bは，ステートメントの接頭語として使われているので，ステートメント・ラベル定数である。LBL_Xは，ステートメント・ラベル変数である。LBL_XにLBL_Aを割り当てた場合は，GO TO LBL_Xというステートメントを与えると，LBL_A ステートメントに制御が渡される。どこか別のところで，このプログラムには，LBL_XにLBL_Bを割り当てるステートメントが含まれているかもしれない。そのような場合は，LBL_Xを参照すると，LBL_Bを参照したのと同じことになる。LBL_X のこの値は，別の値が割り当てられるまでそのままになっている。

ステートメント・ラベル変数は，次のように，LABEL 属性をつけて宣言しなければならない。

```
DECLARE LBL_X LABEL;
```

入口データ

入口データは，入口名とだけ結びつけて使用されるもので，プロシージャーの入口点への参照を可能にする値をもっている。入口データは，入口定数であるときと，入口変数の値であるときがある。

入口定数は，PROCEDUREまたはENTRY ステートメントの接頭語として書かれた入口名として，プログラムの中に出てくる識別名である。これを使用することによって，プロシージャーの入口点を参照することができる。たとえば，

```
DECLARE (SQUARE,CUBE) ENTRY;
          .
          .
          .
     CALL SQUARE;
          .
          .
          .
     CALL CUBE;
```

SQUAREとCUBEは，入口定数として宣言されている。この2つのどちらかの入口定数に参照が行なわれると，SQUARE または CUBEによって指定されたプロシージャー入口点に制御が渡される。

入口変数は，入口定数を参照する識別名である。次の例について考えてみよう。

```
DECLARE EV ENTRY VARIABLE,
   (E1,E2) ENTRY;
      .
      .
      .
EV = E1;
CALL EV;
EV = E2;
CALL EV;
```

EVは，VARIABLE属性によって入口変数として宣言されている。最初のCALLステートメントは，入口定数E1によって表わされる入口点を呼び出している。2番目のCALLは，入口点E2を呼び出している。

事象データ

事象変数は，レコード式入出力操作（またはDISPLAYステートメントの実行）と，その操作を開始したプロシージャーの中にある他のステートメントの実行との間で，ある程度のオーバーラップができるようにするために用いられるものである。

EVENTオプションまたはWAITステートメントの中にEVENT属性を記入することによって，あるいは次の例のようにして明示宣言することによって，変数にはEVENT属性が与えられる。

```
DECLARE ENDEVT EVENT;
```

詳細については，第12章「レコード方式の転送」とセクション4の「表示」の項をみていただきたい。

ロケーター・データ

ロケーター・データには，ポインターとオフセットの2種類がある。

ポインター変数の値は，実際には記憶域内の記憶位置のアドレスであるため，いくつかの異なる記憶位置に記憶域が割り振られている可能性のある変数に対する参照に一定の性質を与えるために使用することができる。

オフセット変数の値は，確保された記憶域の先頭からの相対的な位置を指定するもので，この記憶域のアドレス自体が変ってもそのまま通用する。

ロケーター変数は，次の例のようにして宣言することができる。

```
DECLARE HEADPTR POINTER,
        FIRST OFFSET (AREA1);
```

この例の中で，AREA1は，FIRSTによって指定された記憶位置が含まれる記憶域の名前である。

BASED属性の中，ロケーター修飾記号の左側，またはSETオプションの中にPOINTER属性を記入することによって，変数にPOINTER属性を与えることもできる。

詳細については，第8章「記憶域の制御」をみていただきたい。

エリア・データ

エリア変数は，基底付変数の割振りのために確保される記憶域を記述するために用いられる。エリアは，その中に含まれる割振り場所全体をいっしょに割り当てたり，転送したりすることができる。したがって，1組の基底付の割振り場所を割当てや入出力の1単位として取り扱うことができる一方で，個々の割振り場所はそれぞれ別の存在として識別することもできる。

OFFSET属性またはINオプションの中にAREA属性を記入することによって，あるいは次の例のように明示宣言することによって，変数にAREA属性が与えられる。

```
DECLARE AREA1 AREA(2000),
        AREA2 AREA;
```

確保される記憶域のバイト数は，上の例のAREA1のように，明示的に記述することができる。記述されていないときは，省略時解釈により1000バイトとみなされる。理論上の最大のサイズは16,777,200バイトであるが，実際には，そのプログラムで使用できる記憶域の量によって制約される。

詳細については，第8章「記憶域の制御」をみていただきたい。

データの編成方法

PL/Iでは，データ項目は1個のデータ要素にすることもできるし，それらをいくつかまとめて，配列や構造と呼ばれるデータの集合体を作ることもできる。1つの要素を表わす変数は，<u>要素変数</u>（スカラー変数とも呼ばれる）である。データ要素の集まりを表わす変数は，<u>配列変数</u>または<u>構造変数</u>である。

どのようなタイプの問題データまたはプログラム制御データでも，まとめて配列または構造にすることができる。

配列

データのタイプおよび精度または長さに関して同じ性質をもつデータ要素をいっしょにまとめて配列にすることができる。<u>配列は</u>，その中にある要素がすべて同じ属性をもつ，n－次元の要素の集まりである。配列そのものにだけ，名前が与えられる。配列の中の個々の要素は，配列内における相対的な位置を与えることによって参照される。次のような2つの宣言について考えてみよう。

```
DECLARE LIST (8) FIXED DECIMAL
                        (3);
DECLARE TABLE (4,2) FIXED
                 DECIMAL (3);
```

最初の例では，ＬＩＳＴは，８個の要素からなる１次元の配列で，その中の個々の要素は，３桁の１０進固定小数点項目であると宣言されている。２番目の例では，ＴＡＢＬＥは，８個の１０進固定小数点の要素からなる２次元の配列であると宣言されている。

ＤＥＣＬＡＲＥステートメントの中で配列名の後にある括弧内の数字は，次元属性の指定である。この指定は配列名の後に書かなければならないが，間にブランクを入れても入れなくてもよい。これは，配列の次元の数および各次元の限界（または範囲）を指定するものである。ＬＩＳＴに対しては１つしか限界が指定されていないので，これは１次元の配列である。ＴＡＢＬＥに対しては，カンマで区切って２つの限界が指定されているので，これは，２次元の配列として宣言されたことになる。

次元の限界というのは，その次元の始め（下限）と終り（上限）である。範囲は，上限と下限も含めてその間にある整数の数である。次元に対する限界の指定の中に整数が１つしかないときは，下限は１とみなされる。１次元の配列ＬＩＳＴは，１と８の限界をもち，その範囲は８になる。２次元の配列ＴＡＢＬＥは，１と４，および１と２の限界をもち，その範囲は４と２になる。

次元の下限が１でないときは，上限と下限を表わす２つの数をコロンで結びつけて，その両方を明示的に記述しなければならない。たとえば，

```
DECLARE LIST_A (4:11);

DECLARE LIST_B (-4:3);
```

最初の例の限界は４と１１であり，２番目の例の限界は－４と３である。範囲はどちらも同じであること（つまり，どちらの場合も下限から上限までの間に８個の整数がある）に注意していただきたい。配列の限界と範囲の違いに充分注意していただきたい。２つ以上の配列が関係する配列データを取り扱うとき（第４章「式とデータ変換」のところで説明する）は，それらの配列の限界（範囲だけでなく）は全く同じでなければならない。ＬＩＳＴ，ＬＩＳＴ－Ａ，ＬＩＳＴ－Ｂ の範囲はみな同じであるが，限界は同じではない。

配列の要素を参照するための方法は，配列の限界によって決まる。たとえば，上で宣言されているＬＩＳＴという配列に次のようなデータ項目を割り当てる場合を考えてみよう。

20　5　10　30　630　150　310　70

それぞれの要素は，次のようにして参照されることになる。

参照	要素
LIST (1)	20
LIST (2)	5
LIST (3)	10
LIST (4)	30
LIST (5)	630
LIST (6)	150
LIST (7)	310
LIST (8)	70

ＬＩＳＴという名前の後にある数は，添字である。配列名の後に添字を括弧に入れて（間にブランクを入れても，入れなくてもよい）つけると，配列の中にある特定のデータ項目が識別される。ＬＩＳＴ（４）というような添字付きの名前は，１個の要素を指しており，要素変数である。配列名に添字をつけずに記入すると（たとえば，ＬＩＳＴ），配列全体を指すことができる。この場合，ＬＩＳＴは配列変数である。添字と次元属性の指定の違いに注意していただきたい。後者は，宣言の中に出てくるものであり，それが次元をもっているということと，配列の中にある要素の数を指定するものである。添字は，配列の中にある特定の要素を識別するために，他の参照の中で使用されるものである。

上で宣言したＬＩＳＴ_ＡとＬＩＳＴ_Ｂに同じデータを割り当てると，次のように参照される。

参照	要素	参照
LIST_A (4)	20	LIST_B (−4)
LIST_A (5)	5	LIST_B (−3)
LIST_A (6)	10	LIST_B (−2)
LIST_A (7)	30	LIST_B (−1)
LIST_A (8)	630	LIST_B (0)
LIST_A (9)	150	LIST_B (1)
LIST_A (10)	310	LIST_B (2)
LIST_A (11)	70	LIST_B (3)

２次元の配列として宣言されているTABLEに同じデータを割り当てた場合を考えてみよう。TABLEは，次のような４行２列のマ

トリックス(行列)として表わされる。

TABLE(m,n)	(m,1)	(m,2)
(1,n)	20	5
(2,n)	10	30
(3,n)	630	150
(4,n)	310	70

TABLE内の各要素は，2つの添字をカンマで区切って括弧に入れた添字名によって，参照される。たとえば，TABLE(2,1)は，第2行の最初の項目，この場合は10というデータ項目，を指定することになる。

注：わかりやすく説明するために，TABLEをマトリックスにしたが，これは単に概念的なものであって，各項目が実際に記憶域の中でこのような構成になるわけではない。データ項目は，行を主体とした順序で(つまり，右側の添字が先に変ってゆく)，配列に割り当てられる。たとえば，TABLEへの割当ては，TABLE(1,1),TABLE(1,2)，TABLE(2,1),TABLE(2,2)，……という順に進められる。

配列は，2次元までに限られているのではなく，最高15次元まで宣言することができる。配列の要素を参照するとき，添字付の名前にはその配列の中にある次元の数だけ添字をつけなければならない。

この章では，配列の例として算術データの配列を取り上げてきたが，すべてのタイプのデータを配列にすることができる。ストリングの配列(文字でもビットでも可)も有効であり，ラベル，入口，ファイル，事象，エリア，ロケーターの各データの配列も有効である。

添字としての式

添字付の名前の添字は，定数である必要はない。算術値として有効な値を生ずるものであれば，どのような式を使用してもよい。このような式の計算結果が整数の値でないときは，小数部が無視される。システム／360で実施するときは，添字は，2進整数として内部的に貯えられる。

変数またはその他の式として添字を表わすことは，よくあることである。したがって，TABLE(I,J*K)というような指定法を使えば，I,J,Kの値の変化に応じてTABLE内のいろいろな要素を参照することができる。

配列の断面

添字付の名前の添字を星印に置き換えることによって，配列の断面を参照することができる。星印は，範囲全体を使用することを指定するものである。たとえば，TABLE(*,1)は，TABLEの最初の列のすべての要素を参照しており，TABLE(1,1)，TABLE(2,1),TABLE(3,1)およびTABLE(4,1)からなる断面を指定するものである。添字付の名前TABLE(2,*)は，TABLEの第2行にあるすべてのデータ項目を参照している。TABLE(*,*)は，配列全体を参照するものである。

星印の添字をもつ名前は，1個のデータ要素ではなく，星印の個数に相当する数の次元をもつ配列を表わしていることに注意していただきたい。したがって，このような名前は，要素式ではなく，配列式である。

配列の断面を参照するということは，その配列の中の2つ以上の要素を参照することになるが，このような参照によって指定される要素は，その断面の部分以外の他の要素によって切断されてしまうため，記憶域の中で隣り合っていないこともある。このような断面によって表わされる記憶域は，不連続記憶域と呼ばれる。不連続記憶域の使用には，ある種の制約が適用される。たとえば，レコード変数(レコード式転送ステートメントによってデータが転送される変数)は，連続記憶域(記憶域の中で各データ項目が隣り合っている)の中にあるデータを表わしていなければならない。

構造

同じ性質をもっている必要はないが，互いに論理的な関連をもついくつかのデータ項目をまとめて，構造と呼ばれる集合体を作ることができる。

配列と同様，データの集まり全体を参照するのに用いられる名前が，構造全体に与えられる。ただし，配列とは異なり，構造の中の各要素も，それぞれ名前をもっている。

構造は，上下関係にある名前の集まりである。この関係で最も低い位置にあるのは要素の集まりで，その各々は1個のデータ項目または配列を表わしている。最も上位にあるのは構造名で，要素変数の集まり全体を表わしている。たとえば，次にあげるのは，週給の計算に使用される要素変数の集まりである。

```
LAST_NAME
FIRST_NAME
REGULAR_HOURS
OVERTIME_HOURS
REGULAR_RATE
OVERTIME_RATE
```

これらの変数を集めて構造にし，集合体全体を指す1つの構造名 PAYROLL を与えることができる。

```
                PAYROLL

LAST_NAME       REGULAR_HOURS
                    REGULAR_RATE

FIRST_NAME      OVERTIME_HOURS
                    OVERTIME_RATE
```

PAYROLL を参照すると，PAYROLL という構造内のすべての要素変数が参照される。たとえば，

```
GET DATA (PAYROLL);
```

上の入力ステートメントを与えると，PAYROLL という構造内の要素変数の各々に対してデータを割り当てることができる。

このような集合体をさらに小さな論理的な集合体に小分割すると，都合がよい場合が多くある。上の例の中で，LAST_NAMEと，FIRST_NAMEは論理的な小集合体を作ることができるし，また，REGULAR_HOURSとOVERTIME_HOURS，REGULAR_RATEとOVERTIME_RATEも同じである。構造の中で，このような小集合体に対しても名前をつけることができる。

```
              PAYROLL

NAME       HOURS        RATE

FIRST      REGULAR      REGULAR
LAST       OVERTIME     OVERTIME
```

名前の上下関係には，様々なレベルがあると考えることができる。第1番目のレベルにあるのは，構造名（大構造名とも呼ぶ）であり，それ以下のレベルのものは，下位構造名（小構造名と呼ぶ）である。最も下のレベルにあるのは，要素名（基本項目名と呼ぶ）である。構造の中にある基本項目名が配列を表わすことも可能であるが，この場合の基本項目名は，要素変数ではなく，配列変数を表わしている。

構造の構成は，DECLARE ステートメントの中でレベル番号を使用して指定される。大構造名は，1というレベル番号によって宣言しなければならない。小構造名と基本項目名は，算術的に1より大きいレベル番号（10進整定数でなければならない）によって宣言しなければならない。レベル番号とそれに結びつけられる名前の間は，ブランクをあけて区切ること。たとえば，週給の各項目は，以下のように宣言することができる。

```
DECLARE 1 PAYROLL,
          2 NAME,
            3 LAST,
            3 FIRST,
          2 HOURS,
            3 REGULAR,
            3 OVERTIME,
          2 RATE,
            3 REGULAR,
            3 OVERTIME;
```

注：PAYROLL という構造を実際に宣言するときは，基本項目名LASTとFIRST，および2対のREGULARとOVERTIMEに対してそれぞれ属性を指定する。この例では，記入行を変えたり，記入位置をずらしたりしているが，これは単に読み易くするためのものであって，DECLARE 1 PAYROLL，2 NAME，3 LAST，というように書いてもよい。

PAYROLLは，NAME，HOURS，およびRATEという小構造が含まれる大構造として宣言されている。各々の小構造には，2つの基本項目名が含まれている。プログラマーは，PAYROLLという名前を使って構造全体を参照することもできるし，小構造名を参照することによって構造の一部分だけを参照することもできる。

この宣言の中では，名前の前にはそれぞれ関連するレベル番号がついており，レベル番号と名前の間はブランクで区切られていることに注意していただきたい。次のレベルの番号を選ぶときは，次に大きい整数をつづける必要はない。これらは，単に名前の相対的なレベルを指定するために用いられるものである。レベルnの小構造には，nよりもレベル番号が大きく，しかもその小構造名から，n以下のレベル番号をもつ名前が次に出てくるまでの間にある名前が，すべて含まれることになる。PAYROLLは，次のように宣言することもできる。

```
DECLARE 1 PAYROLL,
          4 NAME,
            5 LAST,
            5 FIRST,
          2 HOURS,
            6 REGULAR,
            5 OVERTIME,
          2 RATE,
            3 REGULAR,
            3 OVERTIME;
```

この宣言は，前記の宣言と全く同じ構造を作り出す。

レベルの数は最大15まで許されるが，レベル番号は最大255まで使うことができる。

大構造名の記述が終了するのは，レベル番号1をもつ別の項目が宣言されたとき，レベル番号をもたない別の項目が宣言されたとき，または DECLARE ステートメントの終了を表わすセミコロンが出てきたとき，である。

レベル番号は，DECLARE ステートメント，および被制御構造のときはALLOCATEステートメントの中でだけ，構造名に対して指定されるものである。構造またはその中の要素を参照するときには，レベル番号は使用しない。

修飾つきの名前

小構造名や構造の要素は，他に同じような名前がなければ，その小構造名や要素名だけを使って参照することができる。しかし，REGULARとOVERTIMEという名前はそれぞれ，PAYROLL に対する構造の宣言の中に2回出てきていることに注意していただきたい。これらの名前を参照するときは，その名前を唯一無二のものにするための何らかの修飾を加えなければ，判別ができない。

PL／I では，このような不明確さを避けるために，修飾つきの名前を使うことができる。修飾つきの名前とは，その名前より高位のレベルにある名前を1つまたは2つ以上使って修飾することにより，唯一無二のものと判別される要素名または小構造名である。PAYROLLの例でいえば，REGULARとOVERTIMEは，HOURS. REGULAR，HOURS. OVERTIME，RATE. REGULAR，RATE. OVERTIMEという修飾名を使うことによって，判別することができる。

修飾つきの名前の中に出てくる様々な名前は，ピリオドで連結される。ピリオドの前後にブランクがあってもよい。修飾は，レベルの順に行なう。つまり，最も高いレベルの名前を最初に書き，最も低いレベルの名前を最後に書くようにする。

大構造名だけは例外であるが，構造の中にある名前は，それが宣言されているプロシージャーの中で唯一無二である必要はない。たとえば，唯一無二の参照にするために，PAYROLL. HOURS. REGULAR という修飾つきの名前が必要なこともある(WORKというような別の構造があって，ここにも小構造 HOURS の中に REGULAR という名前があるとすれば，こちらはWORK. HOURS. REGULAR という名前を用いて判別することができる)．名前を唯一無二のものとして判別ができる範囲で修飾をすればよいので，修飾に用いられる名前を，ある限り全部使用する必要はない。中間にある修飾名を省略することもできる。PAYROLL. NAME. LASTのかわりに，PAYROLL. LASTという名前を使ってもよい。

構造の配列

構造の配列を宣言するために，DECLARE ステートメントの中で，構造名に対して(大構造でも小構造でもよい)次元属性を与えることができる。構造の配列は，その中の要素が同じ名前，レベル，および要素をもつ構造であるような配列である。たとえば，WEATHERという構造が，年間を通じての各月の気象情報を処理するために使用されているとすれば，次のように宣言することができる。

```
DCL 1 WEATHER(12),
      2 TEMPERATURE,
        3 HIGH DECIMAL FIXED(4,1),
        3 LOW DECIMAL FIXED(3,1),
      2 WIND_VELOCITY,
        3 HIGH DECIMAL FIXED(3),
        3 LOW DECIMAL FIXED(2),
      2 PRECIPITATION,
        3 TOTAL DECIMAL FIXED(3,1),
        3 AVERAGE DECIMAL FIXED(3,1);
```

したがって，このような配列が1年間の気象を表わしているときは，プログラマーは，WEATHER(7)と指定することによって，7月の気象データを参照することができる。7月の各部分は，TEMPERATURE(7)，WIND_VELOCITY(7)，PRECIPITATION(7)というようにして参照することができるが，TOTAL(7)とすると，7月中の総雨量が参照される。

3月の最高気温を参照するためのTEMPERATURE. HIGH(3)は，添字をつけた修飾つきの名前である。

構造の配列の中に配列である小構造が含まれているときは，添字をつけた修飾つきの名前がなお一層必要になってくる。たとえば，次のような構造の配列について考えてみよう。

```
DECLARE 1 A (6,6),
          2 B (5),
            3 C,
            3 D,
          2 E;
```

AとBの両方が構造の配列である。この中のデータ項目を識別するためには，3つの名前と3つの添字を使う必要がある場合もある。たとえば，A(1,1). B(2). Cは，Aの中の構造の中にあるBの要素であるCを識別するものである。

添字の順序を変えない限りにおいては，このような参照の中で使われている添字を右または左に移動して，低位または高位のレベルの名前につけることができる。たとえば，上記の構造の配列の場合，A.B.C(1,1,2)およびA(1,1,2).B.Cは，A(1,1).B(2). Cと同じ意味をもつ。すべての添字が最低位または最高位のレベルに移動されていないときは，介在添字をもっているという。したがって，

A．B（1，1，2）．Cは，介在添字をもっているわけである。

構造の配列の中で宣言されている配列は，それが含まれている構造の中で宣言されている次元を受けつぐ。たとえば，構造Aの配列に対する上記の宣言の中では，配列BはAに対して宣言された2つの次元を受けつぐため，3次元の構造になる。Bが唯一無二のものであって修飾を必要としないならば，特定のBを参照するときは3つの添字が必要である（特定のAを識別するために2つ，さらにそのAの中の特定のBを識別するために1つ必要）。

構造の配列の断面

このコンパイラーでは，構造の配列の断面を参照することはできない。つまり，参照の中で星印の表示を使用することはできない。

その他の属性

データ変数に対するキーワード属性（BINARYやDECIMALなど）については，本章の前の節で簡単に説明した。1つのデータ・タイプに個有のものではないような他の属性を使用することもできる。これらの属性の詳細については，I節「属性」をみていただきたい。データ・タイプおよびデータの編成に特に適用される属性は，DEFINED，LIKE，ALIGNED，UNALIGNED，およびINITIAL などである。

DEFINED 属性

DEFINED 属性は，指名されているデータ要素，構造，または配列が，他のデータに割り当てられているものと同じ記憶域を占めることを指定するものである。たとえば，

```
DECLARE LIST (100,100),
    LIST_ITEM (100,100) DEFINED LIST;
```

LISTは，100行100列の2次元の配列である。LIST_ITEMは，LISTの上に定義された，全く同じ配列である。LIST_ITEMの中にある要素を参照することは，LISTの中の対応する要素を参照するのと同じである。

1つのデータ項目を小分割したり，オーバーレーしたりするために，POSITION属性といっしょにDEFINED 属性を使用することができる。たとえば，

```
DECLARE LIST CHARACTER (50),
    LISTA CHARACTER(10) DEFINED LIST,
    LISTB CHARACTER(10) DEFINED LIST
        POSITION(11),
    LISTC CHARACTER(30) DEFINED LIST
        POSITION(21);
```

LISTAは，LISTの最初の10文字を指しており，POSITION(1)とみなされる。LISTBは，LISTの2番目の10文字を指している。LISTCは，LISTの最後の30文字を指している。

新しい配列を構成するために，iSUB変数を使用することによって，配列の一部分を指定するのにDEFINED属性を用いることもできる。iSUB変数は，1から<u>n</u>までの10進整定数として（<u>n</u>は定義されている（被定義）項目の次元の数を表わす）iを指定できる見かけ変数である。iSUB変数の値の範囲は，i番目の次元の下限から上限までである。たとえば，

```
DECLARE A(20,20),
B(10) DEFINED A(2*1SUB,2*1SUB);
```

Bは，配列Aの対角線上にある偶数番目の要素を集めたものから構成されるサブセットである。言葉をかえていえば，B（1）はA（2，2）に対応し，B（2）はA（4，4）に対応する。

<u>不連続記憶域</u>：配列を配列でオーバーレーするためにDEFINED属性を使用すると，不連続記憶域にある配列要素（つまり，記憶域の中で隣り合っていない配列要素）を配列式が参照する可能性が生じる。次の2つの場合には，連続する要素を含む配列式が，不連続記憶域を参照することも可能である。

1. 配列がiSUB定義によって宣言されているとき，iSUB定義によって宣言されている配列の中にある隣り合った要素を参照する配列式は，不連続記憶域を参照することになってもよい（つまり，記憶域の中で隣接していない，オーバーレーされた配列の要素を参照してもよい）。

2. ストリング配列が，それよりも長い要素をもつストリング配列の上に定義されているとき。被定義配列の中にある連続する要素は，基底の配列と被定義配列の各要素の長さの違いによって区切られ，不連続記憶域の中に貯えられているものと考えられる。

LIKE 属性

LIKE 属性は，宣言されている名前に対して，LIKE属性の後

に ある大構造または小構造と同じ構成を与えることを指示するために用いられるものである。たとえば，

```
DECLARE 1 BUDGET,
          2 RENT,
          2 FOOD,
            3 MEAT,
            3 EGGS,
            3 BUTTER,
          2 TRANSPORTATION,
            3 WORK,
            3 OTHER,
          2 ENTERTAINMENT,
        1 COST_OF_LIVING LIKE BUDGET;
```

COST_OF_LIVING に対する上記の宣言は，次のように宣言された場合と同じになる。

```
DECLARE 1 COST_OF_LIVING,
          2 RENT,
          2 FOOD,
            3 MEAT,
            3 EGGS,
            3 BUTTER,
          2 TRANSPORTATION,
            3 WORK,
            3 OTHER,
          2 ENTERTAINMENT;
```

注：LIKE属性は，指定された名前のレベルより低い構造の構成，名前，および属性だけを複写する。指定された名前の次元は複写されない。たとえば，BUDGETが1 BUDGET(12)と宣言されていたとしても，COST_OF_LIVING LIKE BUDGET という宣言によってCOST_OF_LIVINGに次元属性は与えられない。COST_OF_LIVINGに次元を与えるためには，DECLARE 1 COST_OF_LIVING(12) LIKE BUDGETと宣言しなければならない。

小構造名を，LIKE大構造またはLIKE別の小構造というように宣言することができる。また，大構造名を，LIKE小構造またはLIKE別の大構造というように宣言することもできる。

ALIGNEDおよびUNALIGNED属性

システム／360およびシステム／370では，情報は，8ビット単位すなわち8ビットの倍数で貯えられる。8ビットからなる各単位は，バイトと呼ばれる。PL／Iデータが文字形式で貯えられるときは，各文字がそれぞれ1バイトに入る。

バイトは，別々に扱うこともできるし，また，グループにしてフィールドとして扱うこともできる。ハーフワードは，連続する2バイトのグループである。ワードは，連続する4バイトのグループである。ダブルワードは，2ワードからなるフィールドである。記憶装置内のバイトの位置には，0から始まる連続番号がついている。この番号はそれぞれ，対応するバイトのアドレスと考えられる。記憶域の中ではバイトのグループは，グループの左端のバイトによってアドレスされる。

ハーフワードやダブルワードなどの固定長フィールドは，主記憶装置の中では，その情報の単位に対する整数倍の境界に位置していなければならない。その記憶装置アドレスがその単位のバイト数の整数倍になっているとき，情報の単位に対して境界が整数倍になっているという。たとえば，ワード(4バイト)は，そのアドレスが4の倍数になるような記憶位置になければならない。ハーフワード(2バイト)は，2の倍数であるアドレスをもっていなければならず，ダブルワード(8バイト)は，8の倍数であるアドレスをもっていなければならない(図3.1を参照)。

ハーフワード，ワード，およびダブルワードは，その整数倍の境界に位置合せされていない同じ長さのフィールドにくらべて，迅速にアクセスすることができる。このような理由から，ある特定の操作で使用されるデータは，これらの3つの整数境界のいずれかに境界合せするようシステムから要求される。

PL／Iでは，可能な限りの速度で実行ができるような境界にデータを置くことができる。ただしこのような場合は，連続するデータ要素の間に使用されないバイトができ，記憶域の使用量が増すこともあるので，つねに望ましい方法とはいえない。ALIGNEDおよびUNALIGNED属性を使用することによって，プログラマーは，該当する整数境界にデータを貯えるかどうかを選択することができる。

ALIGNEDは，そのデータ・タイプの必要条件に相当する記憶域境界に，データ要素を位置合せするよう指定するものである。このような必要条件は，K節「データ・マップ」の項で指定されている。

UNALIGNEDは，1つの例外をのぞいて，各々のデータ要素は，次のバイト境界上に置くことを指定するものである。唯一の例外は固定長ビット・ストリングの場合で，この場合は，次のビット位置に置かれる。

UNALIGNED属性が指定されていると，コンパイラーは，ある操作でデータの境界合せが必要ならばその操作が行なわれる前に，該当する整数境界にデータを移動するためのコードを生成する。したがって，UNALIGNED属性を使えば記憶域の必要量は減るが，実行時間はふえることもある。

省略時解釈は，要素のレベルで適用される。ビット・ストリング・データ，文字ストリング・データ，および数字データの省略時解釈はUNALIGNEDである。その他のタイプのデータの省略時解釈はすべて，ALIGNED である。

```
         ┌--------------------------------------------------------------------------┐
アドレス | 50000 | 50001 | 50002 | 50003 | 50004 | 50005 | 50006 | 50007 | 50008 |
         └--------------------------------------------------------------------------┘
         | バイト   バイト   バイト   バイト | バイト   バイト   バイト   バイト | バイト

         | ハーフワード       ハーフワード    | ハーフワード       ハーフワード    | ハーフワード

         | ワード                           | ワード                           | ワード

         | ダブルワード                      |                                 | ダブルワード
```

図3.1　固定長フィールドの境界合せを示す主記憶域のセクション

ALIGNEDまたはUNALIGNEDは，要素，配列，または構造変数に対して指定することができる。ある構造に対してどちらかの属性を指定すると，その中に含まれる要素のうち，ALIGNEDまたはUNALIGNEDが明示宣言されていないすべての要素にこの属性を指定したのと同じになる。

次の例は，構造とその要素に対してALIGNEDとUNALIGNEDを宣言した場合の効果を示したものである。

```
DECLARE 1 S,
   2 X BIT(2),          /*UNALIGNED BY
                                  DEFAULT*/
   2 A ALIGNED,         /*ALIGNED
                                  EXPLICITLY*/
     3 B,               /*ALIGNED FROM A*/
     3 C UNALIGNED,     /*UNALIGNED
                                  EXPLICITLY*/
       4 D,             /*UNALIGNED FROM C*/
       4 E ALIGNED,     /*ALIGNED
                                  EXPLICITLY*/
       4 F,             /*UNALIGNED FROM C*/
     3 G,               /*ALIGNED FROM A*/
   2 H;                 /*ALIGNED BY
                                  DEFAULT*/
```

INITIAL 属性

INITIAL 属性は，変数に記憶域が割り振られる時点で，その変数に割り当てる初期値を指定するものである。たとえば，

```
DECLARE NAME CHAR(10) INIT('JOHN DOE');
DECLARE PI FIXED DEC(5,4) INIT(3.1416);
DECLARE TABLE(100,100) INIT CALL(SUBR);
DECLARE A INIT((B*C));
DECLARE X INIT(SQRT(Z));
```

NAMEに記憶域が割り振られるときに，NAMEには'JOHN DOE'という文字ストリング(右側に10個の文字が補充された上で)割り当てられる。PIが割り振られるときは，3.1416という値に初期値化される。どちらの場合もその値は，そのプログラムの間中そのまま変えずにおくこともできるし，実行中に変えることもできる。

3番目の例は，CALLオプションの例である。これは，初期設定を行なうためにSUBRというプロシージャーを呼び出すことを指示している。SUBRの実行中に，所要の値がTABLEに割り当てられる。

4番目の例は，式を含むINITIAL属性を示したものである。これは，BとCの積の値でAを初期値化するよう指定している。

5番目の例は，データ項目を初期値化するために関数参照を用いる例である。

プログラムがロードされるときに割り振られる変数は静的変数と呼ばれるもので，プログラム実行の間中割り振られたままになっており，INITIAL属性の中で指定された値が1度だけ割り当てられる。自動変数の場合は，それが宣言されているブロックが活用化されるたびに割り振られるので，割振りのたびに，指定された初期値が割り当てられる。基底付および被制御変数の場合は，ALLOCATEステートメント(基底付の場合はLOCATEステートメントも)が実行されるときに割り振られるので，割振りのたびに，指定された初期値が割り当てられる。しかし，被制御変数の初期値化は，ALLOCATEステートメントの中で打ち消すことができることに注意していただきたい。

入口定数，ファイル定数，DEFINEDデータ，構造全体，パラメーター(CONTROLLEDパラメーターはのぞく)，または事象データに対して，INITIAL属性を与えることはできない。

注：1つまたは2つ以上の変数を含むCALLオプションまたは式を，静的データに対するINITIAL属性といっしょに使用することはできない。

エリア変数は，割振りの際に，EMPTY組込み関数の値によって自動的に初期値化され，そのあとで，指定されたINITIAL CALLが適用される。エリアは，CALLオプションをもたないINITIAL属性によって初期値化することはできない点に注意していただきたい。

要素変数と同じように，配列に対しても INITIAL 属性を指定することができる。構造の宣言の中では，要素名に対してだけ INITIAL 属性を与えることができる。

配列または構造の配列は，部分的に初期値化することも，全体として初期値化することもできる。初期値化をしない要素は，INITIAL属性の中で省略するか，または星印を使うことによって指定する。たとえば，

```
DECLARE A(15) CHARACTER(13) INITIAL
           ('JOHN DOE',*,'RICHARD ROW',
            'MARY SMITH'),
        B (10,10) DECIMAL FIXED(5)
           INITIAL((25)0,(25)1,(50)0),
        1 C(8),
           2 D INITIAL (0),
           2 E INITIAL((8)0);
```

この例では，Aの1番目，3番目，および4番目の要素が初期値化される。配列の残りの部分は初期値化されない。配列Bは，最初の25個の要素が0に初期値化され，次の25個が1に，最後の50個が0に初期値化されるという形で，完全に初期値化される。括弧内の数字は，初期値化される要素の数を指定する反復因数である。構造Cの場合は，Dは次元(8)を受けついでおり，Dの最初の要素だけが初期値化される。また，Eも次元(8)を受けついでいるが，Eの場合は，すべての要素が初期値化される。

構造の配列を宣言するときにLIKE属性を使用して，すでにその要素が初期値化されている構造と同じ構成にしようとする場合は，この構造の配列の最初の構造だけが初期値化されることに注意しなければならない。たとえば，

```
DECLARE 1 G,
          2 H INITIAL(0),
          2 I INITIAL(0),
        1 J(8) LIKE G;
```

この例では，構造の配列の中でJ(1).HとJ(1).Iだけが初期値化される。

STATIC 配列の場合は，反復因数は10進整定数でなければならない。その他の記憶域クラスの配列の場合は，反復因数は定数，変数，または式にすることができる。

この反復因数と，本章の前の方で述べたストリングの繰返し因数とを混同してはならない。次の例について考えてみよう。

```
DECLARE TABLE (50) CHARACTER (10)
INIT((10)'A',(25)(10)'B',(24)(1)'C');
```

この INITIAL 属性の指定には，反復因数と繰返し因数の両方が含まれている。これは，TABLEの最初の要素が10個のAからなるストリングによって初期値化され，次の25個の要素がそれぞれ，10個のBからなるストリングによって初期値化され，最後の24個の要素がそれぞれ，Cという1桁の文字によって初期値化されることを指定している。ストリング配列に対するINITIAL属性の指定の中では，ストリング定数の前にある括弧に入れられた因数が1個だけのときは，ストリングの繰返し因数とみなされる((10)'A'のような場合)。2個以上あるときは，最初のものが反復因数で，2番目のものがストリングの繰返し因数とみなされる。このような理由から，2つ以上の要素を初期値化するために1個のストリング定数を使用するときは，1というストリング繰返し因数を挿入しなければならない((24)(1)'C'のような場合)。

第4章　式とデータ変換

式というのは，値を表現するものである。1個の定数または変数は式である。演算子および／または括弧を使って結びつけた定数および／または変数も式である。演算子を含む式は，演算式と呼ばれる。演算式の中の定数および変数は，オペランドまたは副式と呼ばれる。

式の例には，次のようなものがある。

```
27
LOSS
A+B
(SQTY-QTY)*SPRICE
```

どのような式も，要素式（スカラー式とも呼ばれる），配列式，または構造式のいずれかに分類することができる。1つの式の中に要素変数，配列変数，および構造変数があってよい。

要素式は，要素の値を表わす式である。この定義には，構造の中にある要素名も含められるし，配列の中の1つの要素を指定する添字つきの名前も要素式である。

配列式は，配列の形をした値を表わす式である。この定義には，次元属性が与えられた構造または構造の一部（小構造または要素）も含まれる。次元属性をもつ大構造または小構造をオペランドとしてもつ式は，さらに構造の配列式として分類することができる。

構造式は，構造の形を与えられた1組の値を表わす式である。そのオペランドは配列ではないが，オペランドに添字をつけることはできる。

次にあげるいくつかの例の中では，各変数は次のように宣言された属性をもつと仮定しよう。

```
DECLARE A(10,10) BINARY FIXED (31),
    B(10,10) BINARY FIXED (31),
    1 RATE, 2 PRIMARY DECIMAL FIXED
            (4,2),
            2 SECONDARY DECIMAL FIXED
            (4,2),
    1 COST(2), 2 PRIMARY DECIMAL FIXED
               (4,2),
            2 SECONDARY DECIMAL FIXED
            (4,2),
    C BINARY FIXED (15),
    D BINARY FIXED (15);
```

次にあげるのは，要素式の例である。

```
C * D
A(3,2) + B(4,8)
RATE.PRIMARY - COST.PRIMARY(1)
A(4,4) * C
RATE.SECONDARY / 4
A(4,6) * COST.SECONDARY(2)
```

これらの式はすべて，その各々のオペランドが要素変数または定数であるから（あるものは配列の式であったり，構造の要素名であったりするが），要素式である。したがって，各々の式は要素値を表わしている。

次にあげるのは，配列式の例である。

```
A + B
A * C - D
B / 10B
RATE + COST
```

上の式はすべて，そのオペランドのうち少なくとも1個が配列変数であるから，配列式である。したがって，どの式もそれぞれ，配列値を表わしている。3番目の例には，2進固定小数点定数10Bが含まれていることに注意していただきたい。最後の例は，構造の配列を表わしている。

次にあげるのは，構造式の例である。

```
RATE * COST(2)
RATE / 2
```

上の式は両方とも，そのオペランドのうち少なくとも1個が構造変数であり，しかも配列のオペランドをもっていないので，構造式である。したがって，これらの式はそれぞれ，構造の値を表わしている。

式の用い方

1個の定数または1個の変数である式は，プログラムの中ではどこでも自由に書くことができる。しかし，多くのPL/Iステートメントの統語規則では，式を評価した結果が正しい値になるものであれ

ば，演算式をそのステートメントの中に書くことが許されている。

本書で使用されている統語の記述の中で，修飾されていない〃式〃という語は，要素式，配列式，または構造式を指している。式の種類が制約されている場合は，その制約が注記してある。たとえば，統語規則の中に〃要素式〃という語があるときは，配列式も構造式も使えないということを表わしている。

注：演算式は，様々なPL／Iステートメントの中に書くことができるが，これが一般に最も多く出てくるのは，次のような形式の割付けステートメントの場合である。

```
A = B + C;
```

割当てステートメントは，PL／Iキーワードをもっていない。割当て記号(=)は，右辺の式（B ＋ C）の値を左辺の変数（A）に割り当てることを意味している。この章では，わかりやすいように，式のいくつかの例を割当てステートメントの中で示してある。

データ変換

演算式

演算式は，1つまたは2つ以上の単一演算から構成されている。単一演算は，接頭演算子演算（単一オペランドの前に演算子がある）または挿入演算子演算（2つのオペランドの間に演算子がある）のどちらかである。挿入演算子演算の2つのオペランドは，操作が行なわれるとき，通常は同じデータ・タイプになっていなければならない。

PL／I では，式の中の演算のオペランドは，必要に応じて，演算が行なわれる前に共通の表現に自動的に変換される。これらの変換に関する一般的な原理が，本章の後の方の「目標フィールドの属性」のところに示されている。変換される項目の精度と長さを計算するための規則も含めて，特別な場合の細かい規則については，F節「データ変換と式の値の計算」のところをみていただきたい。

データ変換は，主として問題データに限定されている。プログラム制御データについて行なわれる唯一の変換は，オフセット・タイプとポインター・タイプの間の変換である。

1つの式の中に2つ以上の表現を使うことについての制約は，非常に少ない。しかし，このように混用すれば，変換を意味することをよく心に留めておかなければならない。実行時に変換が行なわれれば，実行の速度が下がることになる。また，注意が足りないと，変換の結果，精度が失なわれたり，予期しない結果が出たりすることもある。したがって，表現を混用した式はできる限り避けるべきであるし，どうしても使用するときは，プログラマーはそれに関連する変換の規則を充分に理解しておくべきである。

割当て

式を評価する際に行なわれる変換のほかに，データ項目（または式を評価した結果）が，割り当てられる項目の属性と異なる属性をもつ変数に割り当てられるときにも，変換が行なわれる。このような変換の規則は，2，3の例外をのぞいて，演算式を評価する際の変換の規則と同じである。

ストリーム式入出力が行なわれるときにも，変換が行なわれる（第11章を参照のこと）。これは，そのデータが変換される属性をもつ一時的な記憶位置への割当て（入力の直後または出力の直前に行なわれる）の結果としての変換とみなすことができる。

このほかにも，変換を引き起す状況がいくつかある。これらを網羅したリストが，F節に記載されている。

問題データの変換

問題データに対して行なわれる変換には，タイプの変換と算術的変換の2種がある。

タイプの変換は，次のような4つのタイプの問題データの間で行なわれる変換である。

文字ストリング ― CHARACTER属性をもつデータ

ビット・ストリング ― BIT属性をもつデータ

数字―ピクチャー文字AおよびXのどちらも含まれていないPICTURE 属性をもつデータ

コード化算術 ― FIXEDまたはFLOAT，DECIMALまたはBINARY，REALまたはCOMPLEX，および精度の属性をもつデータ

(厳密にいえば，数字データは単に算術データの特殊なケースであるが，変換の規則を表わす目的で，一応別のタイプの表現とみなすことにする。)

算術的変換は，コード化算術形式の中で行なわれる変換 ― 固定小数と浮動小数の間のスケールの変換，10進数と2進数の間の基数の変換，実数と虚数の間のモードの変換，および精度の変換 ― である。

タイプの変換の例として，算術式を評価するときに行なわれる，ビット・ストリングからコード化算術表現への変換がある。ビット・ストリングは，符号のない2進整数とみなされ，そのストリングのビット・パターンによって表わされる正の2進数値に相当する値をもつ，FIXED BINARY(31,0) REAL という属性に，まず変換される。そのときのストリングの長さが31より大きければ，そのストリングの左端の余分のビットが無視される。

算術変換の例として，算術式の評価中に行なわれる，固定小数点10進数表現から浮動小数点2進数表現への，項目(どちらも実数モード)の変換がある。この項目の値は変らず同じであるが，その基数は10進数から2進数に変わり，そのスケールは固定小数点から浮動小数点に変わる。また，ある値を10進整数として表わすためには，3.32倍の2進整数が必要であるから，精度属性の値は，3.32倍だけ増すことになる。この値は，3.32を掛けたあと，整数に切り上げられる。

ロケーター・データの変換

式の評価および割付けの際に変換することのできる唯一のプログラム制御データのタイプは，OFFSETまたはPOINTER属性をもつロケーター・データである。式を評価するときは(=および¬=比較演算子を用いる比較演算に，ロケーター・データを含めることができる)，オフセットからポインターへの変換だけが行なわれる可能性がある。割当てのときは，オフセットからポインターへの変換およびポインターからオフセットへの変換が行なわれる可能性がある。

組込み関数の用い方

式を評価するとき，および割当ての際に変換を行なわせることができるのと同様に，プログラマーは，必要なときにPL/I の組込み関数を用いて変換を開始することができる。(組込み関数の概念については，第9章「サブルーチンと関数」のところを，また関数の詳細については，G節「組込み関数と擬似変数」のところをみていただきたい。)

関数には次のようなものがある。

CHAR
BIT
FIXED
FLOAT
DECITMAL
BINARY

各関数は，その名前によって暗示される属性にデータを変換する。この関数は，必要とされる，タイプの変換および算術的な変換を行なう。上記の関数のほかに，COMPLEX 組込み関数(2つの実数の引数を1つの複素数値に変換する)と関数 REAL(複素数値から実数部分だけを取り出す)がある。

BIT およびCHAR組込み関数の場合は，プログラマーは，その結果のストリングの長さ属性を指定することができるし，また，FIXED，FLOAT，DECIMAL，およびBINARYの場合は，結果の精度を指定することができる。

データ項目の精度は，PRECISION組込み関数によって制御することができる。

ポインター・タイプとオフセット・タイプの間の変換は，OFFSETおよびPOINTER 組込み関数を用いることによって，プログラマーが開始することができる。

これらの組込み関数によって行なわれる変換の多くは，所要の属性をもつPL/I変数に割当てをすることによって，同じように容易に行なうことができる(COMPLEX組込み関数によって行なわれる変換は例外)。しかし，プログラマーは，変換を行なう目的のためだけに変数を作成するよりも，組込み関数を使用する方が便利であることがわかるであろう。

式の演算

演算式は，1つまたは2つ以上の単一の演算を指定することができる。演算のクラスは，演算に対して指定された演算子のクラスによって決まる。演算のクラスには，算術，ビット・ストリング，比較，および連結の4種類がある。

算術演算

算術演算は，次にあげる演算子のいずれかを用いてオペランドを結びつけることによって指定される演算である。

+ － ＊ ／ ＊＊

正の符号と負の符号は，接頭演算子（＋Aや－Aのように，単一のオペランドの前にあってそれと結びつけられている）または挿入演算子（A＋BやA－Bのように，2つのオペランドの間にあってそれらと結びつけられている）のどちらかとして使用することができる。それ以外の算術演算子は，挿入演算子としてだけ使用することができる。

さらに複雑な式は，このような算術演算をいくつか合わせて組にしたものから構成されている。接頭演算子は，挿入演算子演算のオペランドの前におき，それらと結びつけることができることに注意していただきたい。たとえば，A＊－Bという式の中で，変数Bの前にある負の符号は，Aの値とBの負の値とを乗算することを表わしている。

2つ以上の接頭演算子を1つの変数の前におき，それを結びつけることができる。正の接頭演算子を2つ以上重ねても累積的な効果はないが，負の接頭演算子を2つ連続して与えると，正の接頭演算子を1つ与えたのと同じ効果をもつことになる。

算術演算の結果

式の中のオペランドについて必要な変換が行なわれたあと，算術演算が実行され，結果が得られる。この結果は，式の値そのものであることも，さらにそのあとで演算が行なわれる中間結果であることもある。

次の式について考えてみよう。

A ＊ B ＋ C

A ＊ Bという演算が最初に行なわれて，中間結果が出される。そのあと，（中間結果） ＋ Cという演算を実行することによって，式の値が得られる。

中間結果は，コンパイラーによって作られた一時的な記憶位置に入れておかれ，PL／Iプログラムの中の変数と同じようにして属性が与えられる。その結果がどのような属性をもつかは，2つのオペランド（接頭演算子演算の場合は1つのオペランド）の属性と，関係する演算子によって決まる。この関係については，のちにこの章の「目標フィールドの属性」のところで説明する。

中間結果は，さらにその先で演算が行なわれるときに変換されることもあり，また，式の値は，割り当てられるときに変換されることもある。このような変換の場合も，プログラマーが定義したデータと全く同じ規則にしたがって行なわれる。

組込み関数を用いる演算

PL／Iには，加算，減算，乗算，除算に関して実施面で取り決められた精度の規則を打ち消すことができるような組込み関数が3つある。（組込み関数の概念については，第9章「サブルーチンと関数」のところで説明されており，また関数についての詳細は，G節「組込み関数と擬似変数」のところで説明されている。）

このような関数は，ADD，MULTIPLY，およびDIVIDEである。減算されるオペランドに対して負の値を使用するだけで，ADDを減算に使用することができる。これらの関数を使用するときは，結果の精度といっしょに2つのオペランドを指定する。結果の基数，スケール，およびモードは，式を評価するときに変換の規則に従って定義される。

ビット・ストリング演算

ビット・ストリング演算は，次の演算子のいずれかによって2つのオペランドを結びつけることにより指定される演算である。

¬ ε |

最初の演算子"not"記号は，接頭演算子としてだけ使用することができる。2番目と3番目の演算子"and"記号と"or"記号は，挿入演算子としてだけ使用することができる。（これらの演算子は，ブール代数と同じ働きをもつ。）

ビット・ストリング演算のオペランドは，必要に応じて，演算が行なわれる前にビット・ストリングに変換される。挿入演算子演算の2つのオペランドの現在の長さが同じでないときは，短い方のオペランドの右側にゼロが補なわれて拡張される。

ビット・ストリング演算の結果は，両オペランドの現在の長さ（両オペランドの長さは，変換のあとつねに同じになる）と同じ長さのビット・ストリングになる。

ビット・ストリング演算は，1ビットずつ行なわれる。"not"演算をすると，逆のビットが得られる。つまり，¬1の結果は0で

ある。"and" 演算の結果は，該当するビットが両方とも1のときだけ1になり，これ以外の場合は0になる。"or"演算の結果は，オペランドが両方ともゼロのとき以外は1になり，両方ゼロのときは0になる。下の表は，各々の演算子に対する各々のビット位置の結果である。

A	B	¬A	¬B	A&B	A|B
1	1	0	0	1	1
1	0	0	1	0	1
0	1	1	0	0	1
0	0	1	1	0	0

演算結果がビット・ストリングの値になるような1つの式の中で，2つ以上のビット・ストリング演算を組み合わせて用いることができる。

次にあげる例の中で，オペランドAの値が'010111'Bで，オペランドBの値が'111111'B，オペランドCの値が'110'Bとすれば，次のような結果が得られる。

¬ A　の結果は　'101000'B

¬ C　の結果は　'001'B

C & B　の結果は　'110000'B

A | B　の結果は　'111111'B

C | B　の結果は　'111111'B

A | (¬C)　の結果は　'011111'B

¬((¬C)|(¬B))　の結果は　'110111'B

ブール組込み関数

演算子¬，&および|を用いる"not"，"and"，および"or"演算のほかに，BOOL組込み関数を用いてブール演算を行なうことができる。組込み関数の概念については，第9章「サブルーチンと関数」のところを，また，関数の詳細については，G節「組込み関数と擬似変数」のところをみていただきたい。

比較演算

比較演算は，次のような演算子のいずれかを用いて，2つのオペランドを結びつけることによって指定される演算である。

<　¬<　<=　=　¬=　>=　>　¬>

これらの演算子は，"より小"，"より小でない"，"より小または等しい"，"等しい"，"等しくない"，"より大または等しい"，"より大"，および"より大でない"ということを指定するものである。

比較演算には，4つのタイプがある。

1. 代数比較，これは，内部コード化算術形式の符号付算術値の比較をする。オペランドの基数，スケール，精度，またはモードが異なっているときは，算術演算の規則にしたがって変換される。数字データは，比較の前にコード化算術形式に変換される。複素数オペランドの比較の場合は，=および¬=という演算子だけが有効である。

2. 文字比較，これは，照合順序にしたがって，左から右へ1文字ずつ文字の比較をする。

3. ビット比較，これは，左から右へ1ビットずつ2進数字の比較をする。

4. プログラム制御データ比較，これは，内部コード化形式のオペランドの比較をする。比較演算子=および¬=だけが許される。エリアおよび事象変数の比較をすることはできない。オフセットからポインターへの変換だけを行なうことができる。プログラム制御データに対するオペランドについてこれ以外に不一致があると，エラーになる。

比較のオペランド同志が直接的に互換性をもつものでないときは（つまり，それらのデータ・タイプが別のタイプの比較に適するものであるときは），順位の低い方のオペランドが，もう一方の比較のタイプに合わせて変換される。比較タイプの順位は，(1)算術（最高位），(2)文字ストリング，(3)ビット・ストリングの順である。したがって，たとえば，ビット・ストリングを固定小数10進数値と比較する場合は，10進数値との間で算術比較をするために固定小数2進数に変換される（10進数値も固定小数2進数に変換される）。

比較演算の結果はつねに，長さ1のビット・ストリングである。その関係が真のときは，その値は'1'Bになり，関係が偽のときは'0'Bになる。

比較演算は，次のような形式のIFステートメントの中で使われることが最も一般的である。

```
IF A = B
      THEN 真のときの処置
      ELSE 偽のときの処置
```

A ＝ Bという式を評価すると，結果は'1'Bまたは'0'Bのどちらかになる。その値の如何によって，IFステートメントのTHEN部分またはELSE部分が実行される。

しかし，比較演算はIFステートメントに限られるわけではなく，次のような割当てステートメントも有効である。

```
X = A < B;
```

この例では，AがBより小さいときは，'1'Bという値がXに割り当てられ，それ以外の場合は，'0'Bという値が割り当てられる。同様に，次のようなステートメントも有効である。

```
X = A = B;
```

最初の記号(＝)は割当て記号であり，2番目の記号(＝)は比較演算子である。AがBと等しいときはXの値は'1'Bになり，AがBと等しくないときはXの値は'0'Bになる。

連結演算

連結演算は，次のような連結記号を使っていくつかのオペランドを結びつけることによって指定される。

```
||
```

これは，左側のオペランドの最後の文字またはビットのすぐ後に，(間に文字またはビットを介在させずに)，右側のオペランドの最初の文字またはビットをつづけるという形で，オペランドを結合することを指定するものである。

連結演算は，ストリング(文字・ストリングまたはビット・ストリングのどちらか)に対してだけ行なうことができるので，連結演算を指定するとストリング・タイプに変換されることもある。どちらかのオペランドが文字または10進数の場合は，文字ストリングの結果を出すために変換が行なわれる。そのほかに，オペランドがビットと2進数の場合，または両方が2進数の場合は，ビット・ストリングの結果を出すために変換が行なわれる。

連結演算の結果は，以下の通りである。

ビット・ストリング：2つのビット・ストリングのオペランドの長さの合計と等しい長さをもつビット・ストリング

文字ストリング：2つの文字ストリングのオペランドの長さの合計と等しい長さをもつ文字ストリング

連結演算でオペランドの変換が必要な場合は，その結果は，そのオペランドが変換される文字ストリングの長さによって決まる。たとえば，Aが'010111'B，Bが'101'B，Cが'XY,Z'，Dが'AA/BB'という定数の属性と値をそれぞれもっているとすれば，次のような結果になる。

```
    A||B  の結果は '010111101'B
 A||A||B  の結果は '010111010111101'B
    C||D  の結果は 'XY,ZAA/BB'
    D||C  の結果は 'AA/BBXY,Z'
    B||D  の結果は '101AA/BB'
```

最後の例では，連結が行なわれる前に，ビット・ストリング'101'Bが文字ストリング'101'に変換されていることに注意していただきたい。その結果は，8桁の文字ストリングになる。

各種演算の組合せ

1つの演算式の中で，タイプの異なる演算をいくつか組み合わせることができる。どのような組合わせでも使用することができる。たとえば，次の割当てステートメントの中で示されている式は有効である。

```
RESULT = A + B < C ε D;
```

式の中の各々の演算は，そのタイプの演算の規則にしたがって評価されるが，データ変換が必要であれば，演算が行なわれる前に変換される。

上にあげた変数が次のように宣言されているとすれば，

```
DECLARE RESULT BIT(3),
        A FIXED DECIMAL(1),
        B FIXED BINARY (3),
        C CHARACTER(2), D BIT(4);
```

- Aの10進数値は，2進数に変換される。
- AとBを加える2進数加算が行なわれる。

- この2進数の結果が，変換されたCの2進数値と比較される。

- 比較の結果のビット・ストリングが，ビット・ストリングDの長さにまで拡張され，"and" 演算が実行される。

- "and"演算の結果（長さ4のビット・ストリング）が，変換されずに，しかし右側で切り捨てられて，RESULTに割り当てられる。

この例の式は，演算が1つ1つ評価されてゆく順に左から右に記述されている。このような記述の仕方は，あくまでもこの式の場合のことであって，むしろ通常は，式の中にある演算子の優先順位によって評価の順序が決まる。

演算子の優先順位

式を評価するときの演算子の優先順位は，次の通りである。

```
**   接頭演算子+   接頭演算子−   ¬   （最高位）
*  /                                  |
挿入演算子+   挿入演算子−             |
||                                    |
<  ¬<  <=  =  ¬=  >=  >  ¬>           |
ε                                     ↓
|                                 （最下位）
```

最高位の優先順位の演算子が同じ式の中に2つ以上あるときは，これらの演算子は，右から左の順に評価される。つまり，右端の指数演算または接頭演算子が最初に評価される。その左側にある指数演算または接頭演算子が次に評価される最高位の優先順位のものである。

最高位以外の演算子の場合は，同じ優先順位をもつ演算子が1つの式の中に2つ以上あるときは，これらの演算子の順位は，左から右の順である。

次のような割当てステートメントの中で式が評価される順序は，演算子の優先順位によることに注意していただきたい。

```
RESULT = A + B < C ε D;
```

この順序は，この式の各要素を次のように括弧に入れた場合と同じである。

```
(A) + (B)
(A + B) < (C)
((A + B) < C) ε (D)
```

式の評価の順序（したがって，その値も）は，括弧を使うことによって変えることができる。たとえば，前記の式は次のように変えることもできる。

```
(A + B) < (C ε D)
```

この式を順序にしたがって評価すると，長さ1のビット・ストリング（比較演算の結果）になる。このような式を評価するときは，括弧で囲まれた式が先に評価されて1つの値が出され，そのあとで括弧外の演算子との関係が考慮される。しかし，上の例のように括弧の中に入った式が2つある場合にどちらの式を先に評価するか，ということについての規則は，この言語では決まっていない。

Aの値が固定小数点2進数に変換され，次に加算が行なわれて，固定小数点2進数の結果（結果_1）が得られる。Cの値がビット・ストリングに変換され（このような変換が有効であれば），"and" 演算が行なわれる。

この時点で，式は次のようになる。

```
結果_1 < 結果_2
```

結果_2が2進数に変換されてから，代数比較が行なわれ，式全体の結果（ビット・ストリング）が出る。

演算子の優先順位は，オペランド（または副オペランド）の中でだけ定義されるものであって，式全体についてその通りであるとは限らない。次の例について考えてみよう。

```
A + (B < C) ε (D || E ** F)
```

この場合，各演算子の優先順位は，連結演算の前に指数演算を行なうことを指定しているだけであって，他のオペランド（A + (B < C)）の評価との関係で，演算の順序を指定しているのではない。

どのような演算式も（接頭演算式をのぞく），最終的には1つの挿入演算に還元されるものでなければならない。その演算のオペランドと演算子によって，式全体の結果の属性が決まる。たとえば，演算の組合わせについての最初の例（括弧を含まないもの）では，"and" 演算子が，最終的な挿入演算の演算子である。この場合，

式を評価した結果は，長さ4のビット・ストリングである。2番目の例では，（括弧を使用しているので）最終的な挿入演算の演算子は，比較演算子であり，その評価の結果は，長さ1のビット・ストリングになる。

一般に，式の中で括弧が用いられていない限り，最も低い優先順位の演算子によって，最終的な演算のオペランドが決まる。たとえば，

```
A + B ** 3 || C * D - E
```

この例では，最終的な演算が次のようになることを，連結演算子が示している。

```
(A + B ** 3) || (C * D - E)
```

これを評価すると，文字ストリングの結果が得られる。

同様にして，副式を分析することができる。式の2つのオペランドは，次のように定義することができる。

```
A + (B ** 3)

(C * D) - E
```

関数参照のオペランド

式のオペランドは，定数，要素変数，配列変数，または構造変数のいずれであってもよい。以下の割付けステートメントで示されているように，計算の結果の値を表わす式をオペランドにしてもよい。

```
A = B * SQRT(C);
```

この例では，SQRT(C)という式は，Cという値の平方根に等しい値を表わしている。このような式は，関数参照と呼ばれる。

関数参照は，名前と，通常は，1つまたは2つ以上の変数，定数，またはその他の式を括弧に入れたリストから構成される。この名前というのは，リストによって表わされているデータに対して特定の計算を行ない，その計算結果を関数参照の位置に置き換えるためのコーディングのブロックの名前である。

上の例で，Cが16という値をもつと仮定しよう。SQRT(C)という関数参照が与えられると，16の平方根を計算し，4という値で関数参照を置き換えるためのコードが実行される。実際には，この割当てステートメントは次のようになる。

```
A = B * 4;
```

関数参照の中の名前によって表わされるコードは，関数と呼ばれる。SQRTという関数は，PL/Iの組込み関数の1つである。組込み関数は，多くの種々の操作を行なうためのもので，PL/I言語の一部である。各関数に関するくわしいことは，G節「組込み関数と擬似変数」のところをみていただきたい。さらに，プログラマーは，他の目的の関数を書くことができるようになっており（第9章「サブルーチンと関数」のところで説明する），それらの関数の名前を関数参照の中で使用することができる。

関数参照の使用は，演算式のオペランドだけに限られているわけではない。関数参照自体が式であり，式が使用できるところならばどこでも使用することができる。一般に，変数が受入れ側のフィールドを表わしている場合（割当て記号の左側など）は，そこに関数参照を使用することはできない。

しかし，擬似変数として使用できる組込み関数もいくつかある。擬似変数は，受入れフィールドの中で使用される組込み関数である。次の例について考えてみよう。

```
DECLARE A CHARACTER(10),
        B CHARACTER(30);
SUBSTR(A,6,5) = SUBSTR(B,20,5);
```

この割当てステートメントの中で，SUBSTR組込み関数名は，普通の関数参照の中でも，また擬似変数としても使用されている。

SUBSTR組込み関数は，指名されたストリングから指定の長さのサブストリングを抽出するものである。擬似変数として用いられた場合は，受入れフィールドとして指名されたストリングの中の位置を表わしている。

上の例では，ストリングBの20桁目の文字から始まる長さ5文字のサブストリングが，ストリングAの最後の5文字に割り当てられる。つまり，Aの最後の5文字が，Bの20から24桁目の文字と置き換えられるわけである。Aの最初の5文字とBのすべての文字は，変らない。

G節「組込み関数と擬似変数」のところで，擬似変数として使用できるすべての組込み関数をとり上げて説明してある。プログラマー作成の関数は，擬似変数として使用することはできない。

目標フィールドの属性

変換または式演算の目標フィールドは，変換または演算の結果が割り当てられる受入れ側のフィールドである。ここでは，このような目標フィールドの属性を決定するための原理について取り上げる。詳しい規則については，F節「データ変換と式の値の計算」のところで取り上げて説明する。

直接的な割当ての場合，次のようなステートメントで

```
A = B;
```

変換が絶対に必要とすれば，目標フィールドは，割当て記号の左側にある変数（この場合はA）である。しかし，式の評価のとき，目標フィールドは，コンパイラーが作った一時的な記憶位置であることが多い。

次の例について考えてみよう。

```
DECLARE A CHARACTER(8),
        B FIXED DECIMAL(3,2),
        C FIXED BINARY(10);
A = B + C;
```

B＋Cという式を評価するとき，およびその結果を割り当てるときは，次のように4種類の目標フィールドがある。

1. Bを2進数に変換したものが割り当てられる，コンパイラー作成の一時域。

2. 加算の結果の2進数が割り当てられる，コンパイラー作成の一時域。

3. 2進数の結果をそれに相当する固定小数点10進数に変換したものが割り当てられる，コンパイラー作成の一時域。

4. 結果の最終的な行先であるA。固定小数点10進数で表わされた値をそれに相当する文字ストリングに変換したものが，これに割り当てられる。

最初の目標フィールドの属性は，もとになるフィールド(B)の属性，演算子，およびその他のオペランド（算術的な挿入演算子の1つのオペランドが2進数のときは，もう一方は評価の前に2進数に変換される）の属性から決められる。2番目の目標フィールドの属性は，もとになるフィールド（Cおよび変換されたBの表現）の属性から決められる。3番目の目標フィールドの属性は，一部分はもとになるフィールド（2番目の目標）から，一部分は最終的な目標フィールド(A)の属性から決められる。（算術的な2進数の表現は，文字ストリングに変換できるようにするためにはその前に10進数表現に変換しなければならないので，最終的な目標フィールドから決められる唯一の属性はDECIMALである。）4番目の目標フィールド(A)の属性は，DECLAREステートメントから知ることができる。

式が評価されるとき，目標フィールドの属性は通常，一部分はもとになるフィールドから，一部分は実行される演算から，またさらに一部分は2番目のオペランドの属性からとられる。何らかの想定が行なわれることもあるし，実施面での制約（たとえば，最大の精度など）や約束事が何らかの形で存在することもある。式が評価されたあと，さらに結果が変換されることもある。この場合，目標フィールドの属性は通常，もとになるフィールドとは無関係である。

変換にはつねに，もとになるデータ項目と目標のデータ項目とが関係する。つまり，もとになる値の表現と変換された値の表現が関係するわけである。もとになるデータ項目と目標のデータ項目双方の属性のすべてが，コンパイル時にわかっているか，または省略時解釈によって与えられる。

もとになる値によってその属性が決まる可能性のある中間結果を変換に含めることも可能である。たとえば，文字ストリングから算術形式に変換するには，中間的な変換が必要であり，したがって，最終的な変換が完了する前に中間結果が出ることになる。しかし，このような場合の最終の目標フィールドの属性は，つねにもとになるデータ項目から決められるので，変数の値とは無関係である。

定数もまた属性をもつことをよく理解しなければならない。1.0という定数は，1，'1'B，'1'，1B，1E0などという定数とは異なっている。コンパイル時または実行時に定数を変換することができるが，どちらの場合もその規則は同じである。

配 列 式

配列式は，1つの配列変数，または少なくとも1つの配列オペランドを含む式である。配列式にも，演算子（接頭，挿入のどちらでも），要素変数，および定数を含めることができる。

配列式を評価すると，結果は配列になる。配列に対して行なわれるすべての演算は，1つ1つの要素について行を主とした順序で行なわれる。したがって，配列式の中で参照されるすべての配列は，次元の数が同じで，しかも各次元の限界も全く同じでなければならない。

配列オペランドといっしょに比較演算子を使用してもよいが，ＩＦステートメントのＩＦクローズの中で配列オペランドを使用することはできない。ＩＦステートメントは，１つの真，偽いずれかの結果をテストするものであるから，ＩＦクローズの中で使用してよいのは，要素式だけである。

注：配列式は，通常，数学でいう行列式と同じではない。

接頭演算子と配列

配列に対する接頭演算子演算の結果は，同じ境界をもつ配列であり，その中の要素はそれぞれ，もとの配列の各要素に対して行なわれた演算の結果になる。たとえば，

Ａが右のような配列のときは	5	3	−9
	1	2	7
	6	3	−4
−Ａは右のような配列になる	−5	−3	9
	−1	−2	−7
	−6	−3	4

挿入演算子と配列

１つのオペランドとして配列変数を含む挿入演算は，もう一方のオペランドとして要素，別の配列，または構造を含むことができる。

配列と要素の演算

要素と配列が挿入演算子によって結びつけられている演算の結果は，もとの配列と同じ限界をもつ配列になり，その中の各々の要素は，もとの配列の中の対応する要素と１つの要素との間で行なわれた演算の結果になる。たとえば，

Ａが右のような配列のときは	5	10	8
	12	11	3
Ａ＊３は右のような配列になる	15	30	24
	36	33	9

配列と要素の演算における要素を，同じ配列の中にある要素にすることができる。たとえば，Ａ＊Ａ（２，３）という式の結果は，Ａ（２，３）の値が３なので，前記の配列Ａの場合と同じ結果になる。

次の割当てステートメントについて考えてみよう。

```
A = A * A(1,2);
```

再び，Ａに対して上記の値を用いれば，新たにＡに割り当てられる値は次のようになる。

50	100	800
1200	1100	300

Ａ（１，２）に対するもとの値（このときは１０である）は，評価の際にＡの最初の２つの要素に対してだけ用いられることに注意していただきたい。式の結果がＡに割り当てられ，Ａの値が変わるので，Ａ（１，２）の新しい値がその後の演算に用いられる。最初の２つの要素は，１０，すなわちＡ（１，２）のもとの値で乗算される。その他の要素はすべて，１００，すなわちＡ（１，２）の新しい値で乗算される。

配列と配列の演算

１つの挿入演算子によって２つの配列が連結されている場合は，２つの配列は全く同じ限界をもつものでなければならない。その結果は，もとになる配列の限界と同じ限界をもつ配列になる。もとになる２つの配列の対応する要素同志に対して演算が行なわれる。

２つの配列は全く同じ次元数のものでなければならず，また，対応する各次元の下限および上限も全く同じでなければならない。たとえば，Ｘ（１０，６）と宣言された配列の限界は，Ｙ（２：１１，３：８）と宣言された配列の限界と同じではない（対応する次元に対する範囲は同じであるし，要素の数も同じではあるが）。

次にあげるのは，配列の挿入演算式の例である。

Ａが右のような配列で	2	4	3
	6	1	7
	4	8	2
Ｂが右のような配列のときは	1	5	7
	8	3	4
	6	3	1

A＋Bは右のような配列になり

3	9	10
14	4	11
10	11	3

A＊Bは右のような配列になる

2	20	21
48	3	28
24	24	2

配列と構造の演算

挿入演算子によって配列と構造が結びつけられている演算の結果は，その配列と全く同じ限界をもち，その構造と全く同じ構造をもつ構造の配列になる。

たとえば，次のような宣言が与えられているとすれば，

```
DECLARE 1A, 2 B, 2 C,
        X(2),
        Y(2) LIKE A;
```

次の割当てステートメントは，

```
Y = X + A;
```

有効である。これは次のものに相当する。

```
Y.B(1) = X(1) + A.B;
Y.C(1) = X(1) + A.C;
Y.B(2) = X(2) + A.B;
Y.C(2) = X(2) + A.C;
```

配列式におけるデータ変換

配列式についてのこの説明の中であげてある例では，単一の算術演算だけが示されている。演算を組み合わせる際の規則およびオペランドのデータ変換の規則は，要素演算の場合と同じである。

構　造　式

構造式は，少なくとも1つの構造オペランドを含み，しかも配列オペランドは含まない，という1個の構造変数または式である。要素変数と定数は，構造式のオペランドにすることができる。構造式を評価すると，その結果は構造になる。構造オペランドは，大構造名または小構造名のどちらであってもよい。

比較演算子に対して構造のオペランドを用いてもよいが，IFステートメントのIFクローズの中で構造オペランドを使用することはできない。IFステートメントは，1つの真，偽の結果をテストするものであるから，IFクローズの中で使用できるのは，要素式だけである。

構造に対して行なわれるすべての演算は，要素の1つ1つについて行なわれる。BY NAMEによる割当ての場合（次のページを参照）をのぞいて，構造式の中に出てくる構造変数はすべて，同じ構造になっていなければならない。

同じ構造であるということは，その構造内の小構造が同じ形になっており，その中に含まれる要素と配列の数が同じでなければならないこと，およびその構造内（小構造があればその小構造の中でも）の要素と配列の位置が同じでなければならないことを意味している。対応する位置にある配列は，同じ限界をもつものでなければならない。名前は同じである必要はない。正しい変換が行なわれるためには，対応する要素のデータ・タイプが同じでなければならない。

接頭演算子と構造

構造に対して接頭演算子演算をすると，その結果は同じ形式の構造になり，その各々の要素は，もとの構造の各々の要素に対して実行された演算の結果になる。

注：構造には，多くの異なるデータ・タイプの要素を含めることができるので，構造変数（大構造名，小構造名のどちらでも）によって表わされるすべての要素に対して演算を正しく実行することができない限り，構造式の中の接頭演算子演算は無意味になる。

挿入演算子と構造

1つのオペランドとして構造変数を含む挿入演算は，もう一方のオペランドとして要素または別の構造をもつことができる。

構造式の中の構造オペランドは，大構造名である必要はない。小構造名は，それがどのようなレベルのものであっても，構造変数である。したがって，M．Nが大構造Mの中の小構造であれば，次にあげるものは，構造式である。

```
M.N ε '1010'B
```

構造と要素の演算

演算の一方のオペランドが構造で，もう一方が要素の場合は，構造の中にある各々の要素について1つずつ演算を連続して行なうのと同じである。各々の副演算では，構造内の1つの要素と単一の要素が用いられる。

次の構造について考えてみよう。

```
1 A,
  2 B,
    3 C,
    3 D,
    3 E,
  2F,
    3 G,
    3 H,
    3 I;
```

Xが要素変数であれば，A * Xは次のものと等しい。

```
A.C * X
A.D * X
A.E * X
A.G * X
A.H * X
A.I * X
```

構造と構造の演算

1つの演算が2つの構造オペランドをもっているときは，対応する各々の対に関して1つずつ要素演算を連続して行なうのと同じである。たとえば，Aが，前の例で示した構造であり，Mが次のような構造をもっているとすれば，

```
1 M,
  2 N,
    3 O,
    3 P,
    3 Q,
  2 R,
    3 S,
    3 T,
    3 U;
```

A || M は次のものと等しくなる。

```
A.C || M.O
A.D || M.P
A.E || M.Q
A.G || M.S
A.H || M.T
A.I || M.U
```

BY NAME による構造の割当て

構造式のオペランドが同じ構造になっていなければならないという規則に対する唯一の例外は，BY NAME オプションをもつ割当てステートメントの中に構造式がある場合である。

BY NAME は，構造の割当てステートメントの最後のところに，カンマをつけてその後に書く。以下に示すのは，その例である。

次の構造と割当てステートメントについて考えてみよう。

```
1 ONE,             1 TWO,            1 THREE,
  2 PART1,           2 PART1,          2 PART1,
    3 RED,             3 BLUE,           3 RED,
    3 ORANGE,          3 GREEN,          3 BLUE,
  2 PART2,             3 RED,            3 BROWN,
    3 YELLOW,        2 PART2,          2 PART2,
    3 BLUE,            3 BROWN,          3 YELLOW,
    3 GREEN;           3 YELLOW;         3 GREEN;

   ONE = TWO, BY NAME;
   ONE.PART1 = THREE.PART1, BY NAME;
   ONE = TWO + THREE, BY NAME;
```

最初の割当てステートメントは，次のものと同じである。

```
ONE.PART1.RED = TWO.PART1.RED;
ONE.PART2.YELLOW = TWO.PART2.YELLOW;
```

2番目の割当てステートメントは，次のものと同じである。

```
ONE.PART1.RED = THREE.PART1.RED;
```

3番目の割当てステートメントは，次のものと同じである。

```
ONE.PART1.RED = TWO.PART1.RED
                + THREE.PART1.RED;
ONE.PART2.YELLOW = TWO.PART2.YELLOW
                   + THREE.PART2.YELLOW;
```

BY NEME オプションは，割当てステートメントの中にだけ書くことができる。これは，両方の構造に共通な名前をもつ要素に対してだけ，構造の要素の割当てを行なうことを示すものである。割当てステートメントの中で指定される最高レベルの修飾語をのぞいて，修飾する名前はすべて同じでなければならない。

BY NAME オプションをもつ割当てステートメントの中に演算式があるときは，それぞれの構造の中でその名前が宣言されている要素に対してだけ，演算と割当てが行なわれる。上記の3番目の割当てステートメントの場合は，TWO の PART2 の中に要素名として GREEN は出ていないので，ONE.PART2.GREEN および THREE.PART2.GREEN に対して，演算は行なわれない。

例外条件

データの変換中に起こり得るPL／Iの例外条件には，SIZE，CONVERSION，およびSTRINGSIZEの3つがある。（条件の概念は，第14章「例外条件の処理とプログラム・チェックアウト」のところで説明されており，また，諸条件の詳細については，H節「ON条件」のところで説明されている。）

SIZE条件は，算術値の左側から有効数字が失なわれたときに起こる。この条件は，式の中で変換が行なわれるとき，または式の結果を割り当てるときに起こり得るものであって，文字ストリングまたはビット・ストリングに変換されるときには，値が切り捨てられても起らない。編集指示転送でEまたはF形式に変換するときに，指定されたフィールドの幅がリスト項目の値を収容できない場合にもこの条件が起こる。SIZE接頭語の有効範囲内で条件が起ったときにだけ割込みを起こすように，通常，SIZE条件は禁止状態にされる。

CONVERSION条件は，もとになるフィールドの中に，行なわれる変換に対して有効でない文字が含まれているときに，起こる。たとえば，算術形式に変換される文字ストリングの中に算術定数で許される文字以外の文字が含まれている場合や，ビットに変換される文字ストリングに0と1以外の文字が含まれている場合などに，CONVERSIONが起こる。無効な文字があると，その各々について1度ずつCONVERSION条件が起こるので，無効な文字が2つ以上あると，1回の変換操作で数回の割込みが起こる。CONVERSION条件は通常，解禁されているので，条件が起ったときは割込みが起こる。この条件は，NOCONVERSION接頭語を使って禁止状態にすることができる。このような場合は，条件が起っても割込みは起らない。

STRINGSIZE条件は，文字またはビット・ストリングが割り当てられる目標フィールドが小さすぎてストリングを収容できないときに起こる。目標フィールドの長さに合うように，文字またはビットの右端が切り捨てられる。STRINGSIZE条件は，STRINGSIZE条件接頭語の有効範囲内でだけ割込みが起こるように，通常は禁止状態にされる。

これらの3つの条件は，式の評価中にも起こり得る。さらに，FIXEDOVERFLOW，OVERFLOW，UNDERFLOW，およびZERODIVIDEの4つの条件も起り得る。FIXED-OVERFLOWとOVERFLOWは，目標フィールドについて宣言された精度を越えたときではなく，実施面で定義された最大精度を越えたときに起こることに注意していただきたい。

第5章　ステートメントの分類

この章では，機能別にステートメントを分類している。機能別のクラスに分けてステートメントがリストされており，各ステートメントの目的とその用法の例が記載されている。

各ステートメントについての詳細な説明は，本章ではなく，J節「ステートメント」をみていただきたい。

ステートメントの種類

ステートメントは，次のようなクラスに分けることができる。

記述ステートメント

入出力ステートメント

データ移動および計算ステートメント

プログラム構成ステートメント

記憶域制御ステートメント

制御ステートメント

例外制御ステートメント

プリプロセッサー・ステートメント

それぞれのクラスの名前は，説明の便宜上つけたものであって，プリプロセッサー・ステートメント以外は，言語の中でこの名前が特別な意味をもっているわけではない。ステートメントの中には，いくつもの働きをもつものもあるので，2つ以上のクラスに属するものもある。

記述ステートメント

PL/I ステートメントが実行されるときは，非常に多くの種類のデータを扱うことができる。算術定数またはストリング定数以外の各データ項目は，プログラムの中では名前によって参照される。PL/I 言語では，参照されるデータ項目の特性（または属性）は，プログラムのコンパイル時にわかっていなければならない。ただし，この規則には2，3の例外がある。たとえば，非 STATIC 項目の場合は，配列の次元の限界，ストリングの長さ，エリアの大きさ，初期値，ある種のファイル属性は，プログラムの実行中に決めることもできる。

DECLAREおよびDEFAULTステートメント

DECLARE ステートメントは，名前の属性を指定するための主要な方法である。プログラムの中で使用される名前は，つねに DECLARE ステートメントの中にある必要はなく，文脈によってその属性を決めることができる場合も多い。属性が明示的に宣言されておらず，しかも文脈からも決められないという場合は，省略時解釈の規則が適用される。省略時解釈の規則には，そのコンパイラーで定義された標準の省略時解釈の規則と，DEFAULT ステートメントを使って特定のプログラム用にプログラマーが定義したもの，とがある。省略時解釈と文脈による決定とをうまく組み合わせれば，DECLAREステートメントを使う必要がない場合もある。

DEFAULT ステートメントを使うことによって，プログラマーは，省略時解釈が適用される以下のような属性に対して制御を加えることができる。

明示宣言された識別名

文脈宣言された識別名

暗示宣言された識別名

ENTRY 属性の中の記述子

内部プロシージャーから返された値

DECLARE ステートメントは，プログラムの文書化という面からも重要な部分である。したがって，プログラマーは，その属性が省略時解釈できるものであっても，また文脈宣言が可能であっても，宣言を自由に用いることができる。DECLARE ステートメントの数に制限はないので，様々な名前のグループに対して，別々の DECLARE ステートメントを使用することができる。このようにすると，修正が容易になり，診断の解読も明確になる。

その他の記述ステートメント

OPEN ステートメントは，ファイル定数に関してある種の属性を指定することができるので，記述ステートメントとして分類されることもある。被制御変数に対する特定の属性を，ALLOCATE ステートメントの中で指定することができる。FORMAT ステートメントは，外部媒体上におけるデータのレイアウト，たとえばページや入力カードなど，を記述するものと考えることもできる。

入出力ステートメント

入出力クラスに属する主要なステートメントは，内部記憶装置と外部媒体との間で実際にデータの転送を行なうものである。その他の入出力ステートメントは，このような転送に影響を与えるものであって，入出力制御ステートメントと考えることができる。

各入出力ステートメントは，ファイルを識別するための関連FILEオプションをつけて使用される。ファイル・オプションは，ファイル定数，ファイル変数，またはファイルの値を返す関数のいずれかであるようなファイル式を指定するものである。

以下のリストの中では，データの転送を引き起すステートメントは，RECORD入出力とSTREAM入出力の2クラスに分けてある。

RECORD 転送ステートメント

- READ
- WRITE
- REWRITE
- LOCATE
- DELETE

STREAM 転送ステート

- GET
- PUT

入出力制御ステートメント

- OPEN
- CLOSE

これらのステートメントに関連のあるステートメントとして，DISPLAY ステートメントがある。

STREAM転送とRECORD転送の間には，2つの重要な相違がある。STREAM 転送では，各データ項目は別々に扱われるが，RECORD 転送は，データ項目の集まり（レコード）を1つの全体として扱う。STREAM 転送では，各項目は，転送されるときに編集され，変換される。一方，RECORD 転送では，編集も変換も行なわれず，外部媒体上のレコードは内部記憶装置内にあるときと全く同じレコードのコピーであるのが普通である。

このような相違があるので，RECORD 転送は，内部表現（2進数や10進数など）で書かれている大きなファイルを処理するのに特に適している。それに対して，STREAM転送は，穿孔データを処理したり，読みやすいデータを作成したり（このような場合は編集が必要）するのに用いられる。

レコード転送ステートメント

READ ステートメントは，内部記憶装置にレコードを直接転送し，それらを処理できるようにする。

WRITE ステートメントは，レコードを出力装置に転送する。

LOCATE ステートメントは，緩衝域内における位置を指し示すポインターをセットすることによって，出力緩衝域の中に変数を入れる記憶域を割り振るものであるが，その際に，このファイル用の緩衝域の中にすでにレコードが入っていれば，先にそれを書き出してから上記のことを行なう。

REWRITE ステートメントは，UPDATE ファイル内の既存のレコードを変更する。

DELETE ステートメントは，UPDATE ファイルから既存のレコードを削除する。

ストリーム転送ステートメント

GETおよびPUT ステートメントで処理することができるのは，順次ファイルだけである。レコードの境界は，通常無視される。データは，外部媒体から入ってきた（GET）か，外部媒体に出てゆく（PUT）個々のデータ項目の流れを考えることができる。

GETおよびPUT ステートメントは，3つのモード（データ指示，リスト指示，または編集指示）のいずれかで，項目のリストを転送することができる。データ指示転送では，データ項目の名前とその値が，外部媒体上に記録される。リスト指示転送では，データは，ブランクまたはカンマによって区切られて定数のリストとして外部的に記録される。編集指示転送では，データは，形式リストにしたがって1文字ずつ処理される文字のストリングとして外部的に記録される。

データ指示転送は，比較的少量の値を読むときや，注釈付のデバッギング用の出力を作る際に非常に有効である。リスト指示入力は，自由形式で穿孔された大量のデータを読むのに適している。編集指示転送は，形式を厳密に制御しなければならないとき，たとえば報告書を作成するときや固定形式で穿孔されたカードを読むときに，使用される。

注：GETおよびPUTステートメントは，FILEオプションを指定するかわりに，STRINGオプションの中でストリング名を指定することによって，内部的なデータ移動にも使用することができる。この機能は，READ SETおよびLOCATEステートメントを用いて，緩衝域への（および緩衝域からの）データ移動にも使用することができるが，これは，実際には入出力操作の一部分ではない。STRINGオプションをもつGETおよびPUTステートメントは，この章の「データ移動と計算ステートメント」の項で説明する。

入出力制御ステートメント

OPENステートメントは，ファイル名をデータ・セットに結びつけ，そのデータ・セットの処理の準備をするものである。また，そのファイルに対して追加の属性を指定することもできる。

OPENステートメントは，つねに指定しなければならないわけではない。まだ開かれていないファイルの名前を指定する入出力転送ステートメントが実行されると，データ転送が行なわれる前に，ファイルは自動的に開かれる。

ファイルに対してINPUTまたはOUTPUT属性を指定するために，OPENステートメントを使用することができる。たとえば，PRINTファイルの場合，各印刷行の長さと1ページ当りの行数は，OPENステートメントの中でPAGESIZEおよびLINESIZEオプションを用いることによってのみ指定することができる。LINESIZEオプションは，装置に転送される物理ブロックの長さを決定するために，PRINT OUTPUT以外のファイルに対しても指定することができる。OPENステートメントは，データ・セットとファイルとを連係するものとして，ファイル名以外の名前を指定する（TITLEオプションの中で）ために使用することもできる。

CLOSEステートメントは，データ・セットをファイルから切り離すものである。プログラムが終了するときにはすべてのファイルが閉じられるので，CLOSEステートメントは，必ずしも指定しなくてもよい。

DISPLAYステートメント

DISPLAYステートメントは，操作卓上にメッセージ（通常は，操作員に対するもの）を書き出すために用いられる。REPLYオプションをつけてこのステートメントを使用すれば，操作員は，コードまたはメッセージをタイプすることによってプログラムと連絡することもできる。REPLYオプションは，操作員がメッセージを確認するまでプログラムの実行を保留しておくだけの目的で使用することもできる。

データ移動と計算ステートメント

内部的なデータ移動とは，指定された変数に式の値を割り当てることである。式は，定数または変数のいずれでもよく，また，これから行なわれる計算を指定する式であってもよい。

内部的なデータ移動のために，また計算を指定するために，最も一般的に用いられるステートメントは，割当てステートメントである。STRINGオプションをもつGETおよびPUTステートメントも，内部的なデータ移動に使用することができる。さらに，PUTステートメントは，これから行なわれる計算を指定することもできる。

割当てステートメント

割当てステートメントは，キーワードをもたないが，割当て記号(=)によって識別される。これは，通常，次の2つの形式のうちどちらかの形式をとる。

A = B;

A = B + C;

最初の形式は，単なる内部的なデータ移動に用いることができる。割当て記号の右側にある変数（または定数）の値が，左側にある変数に割り当てられる。2番目の形式は，演算式を含んでおり，その値が，割当て記号の左側にある変数に割り当てられる。2番目の形式は，データの移動はもちろんであるが，行なうべき計算も指定するものである。

左辺の変数の属性が，式の結果（または変数あるいは定数）の属性と異なっていてもよいので，割当てステートメントを用いて変換または編集を行なうこともできる。

左辺の変数を，配列または構造の名前にすることもできる。つまり，右辺の式の結果が配列または構造の値になってもよい。したがって，単一の項目と同様，データの集合体を移動するために割当てステートメントを用いることができる。

複数割当て

次のような形式のステートメントの中では，割当てステートメントの式の値を2つ以上の変数に割り当てることができる。

A,X = B + C;

このようなステートメントは，B + Cの値が1度だけ計算され，次にAとXに割り当てられる点以外は，単一の割当てと全く同じようにして実行される。

注：BY NAME による構造の割付けに複数割当てを使用する場合は，影響を受ける要素名は，割当て記号の左側にリストされている構造のすべてに共通する要素名だけである。

プログラム編成ステートメント

プログラム編成ステートメントとは，プログラムをセクションに区切って，ブロックおよびグループにするために用いられるステートメントである。このようなステートメントには，PROCEDURE ステートメント，END ステートメント，ENTRY ステートメント，および BEGIN ステートメントがある。

プログラムを適当なブロックに分けると，プログラムの作成とテストが簡素化される。これは，幾人かのプログラマーが1つのプログラムを共同して作成する場合にとくに有効である。また，自動記憶域クラスの動的な割振りは，それが宣言されているブロックに入るときに割り振られるので，この方法を使用すれば，記憶域を有効に使用することができる。

PROCEDURE ステートメント

プロシージャー・ブロックは，PROCEDURE ステートメントおよび関連する END ステートメントによって区切られるが，その主な働きは，指定されたデータに対して行なわれる一連の操作を定義することである。このような一連の操作には名前（PROCEDURE ステートメントのラベル）が与えられ，その名前が知られているところであればどこからでも呼び出すことができる。

どのようなプログラムも，少なくとも1個の PROCEDURE ステートメントと1個の END ステートメントをもっていなければならない。プログラムは，別々に作成されたあとで連係されたいくつかのプロシージャーから構成することができる。プロシージャーの中に，ネストされた他のプロシージャーが含まれていてもよい。これらの内部プロシージャーの中に，局部的に定義された名前として扱われる（他に指定されていなければ）宣言が含まれていてもよい。このような定義は，自分のブロックの外では知られていないので，その外側のプロシージャーの中でこのような名前を参照することはできない。これらの名前と結びつけられる記憶域は，通常，そのような名前が定義されているブロックに入るときに割り振られ，そのブロックから出るときに解放される。

プロシージャーによって定義されている一連のステートメントは，プロシージャー名が知られているところであれば，どこからでも実行することができる。プロシージャーは，CALL ステートメントによって呼び出すこともできるし，式の中にその名前を書くことによっても呼び出すことができる（後者の場合は，関数参照と呼ばれる）。関数参照は，値を計算して，式の評価の中で使用するためにその値を関数参照に戻す。

2つのプロシージャーの間の連絡は，呼出し側のプロシージャーから呼び出される側のプロシージャーに引数を渡したり，呼び出される側のプロシージャーから値を返したり，両方のプロシージャーの中で知られている名前を用いたりすることによって行なわれる。したがって，プロシージャーは，別の地点から呼び出されれば，別のデータを使って働くことができる。RETURN ステートメントを用いて，関数プロシージャーから関数参照に値が返される。

ENTRY ステートメント

ENTRY ステートメントは，あるプロシージャーに対して代替の入口点を設け，場合によっては，その入口点に応じた代替のパラメーター・リストを作って引数を渡すことができるようにするものである。

注：ENTRY ステートメントは，それが入っているプロシージャーへの入口点を指定するものであって，ENTRY 属性とは違うことに注意していただきたい。ENTRY 属性については，第9章「サブルーチンと関数」のところで説明する。

BEGIN ステートメント

開始ブロック（BEGIN ステートメントとそれに関連する END ステートメントとによって区切られているブロック）の中でも，局部的な名前の定義をすることができる。開始ブロックは，その中に含まれるステートメントが，制御の流れという目的からみて1つの存在と考えられることを指定するものである。開始ブロックの最も一般的な用法の1つとして，ON ステートメントのオン・ユニットとして使用する場合があげられる。この場合は，正規の制御の流れを通ってきた結果ではなく，指定の条件が起ったときにだけ実行される。開始ブロックは，ある種の自動記憶域が割り振られるプログラムのセクションを区切るためにも有効である。

各々の開始ブロックは，プロシージャーまたは別の開始ブロックの中にネストしなければならない。

END ステートメント

END ステートメントは，ブロックまたはグループの終りを指示するために用いられる。ブロックまたはグループはいずれも，END ステートメントをもっていなければならない。しかし，END ステートメントは，明示することも，暗示することもできる。つまり，END というキーワードの後に，ネストされたブロックを含むブロックまたはグループのラベルを記入することによって，1つのEND ステートメントを，いくつものネストされたブロックまたはグループに適用することができる。このようにすると，その他のEND ステートメントは，そのラベルを含んでいるものによって暗示されるので，明示的に与える必要がない。END の後にラベルがない場合は，そのステートメントは，1つのグループまたはブロックにだけ適用される。

あるブロックに対してEND ステートメントが実行されると，そのブロックは終了する。しかし，各ブロックはEND ステートメントをもっていなければならないが，これがブロックを終了させる唯一の手段というわけではない。たとえば，RETURN ステートメントを実行することによって，プロシージャーを終了させることもできる（「制御ステートメント」の項を参照）。

グループに対してEND ステートメントを実行したときの効果は，そのグループが反復するものであるかどうかによって異なる。反復するグループの場合は，END ステートメントを実行すると，すべての反復が完了するまで（その前にグループの外に制御が渡されない限り），グループの先頭に制御が戻される。（「制御ステートメント」の項を参照）。反復しないグループの場合は，END ステートメントは，単にグループを区切るだけの働きをして（プログラム論理の中で1つの単位としてグループが扱われるようにする），次のステートメントに制御を渡す。

記憶域制御ステートメント

PL／Iで決められている多くの他の約束事と同じように，記憶域割振りに関する約束事も，プログラマーは打ち消すことができる。多くの変数の場合，記憶域クラスの属性AUTOMATIC が想定される。しかし，変数に対して STATIC を宣言することもできるし（この場合は，そのプログラムの間中割り振られる），CONTROLLEDまたはBASEDを宣言することもできる（この場合は，その割振りは，プログラマーによって明示的に指定することができる）。

ALLOCATEおよびFREEステートメント

ALLOCATE ステートメントは，ブロックの境界とは無関係に，被制御および基底付データに対して記憶域を割り振るために用いられる。被制御配列の限界，被制御ストリングの長さ，および被制御域のサイズは，それらの初期値と同様，ALLOCATE ステートメントが実行されるときに指定することもできる。FREE ステートメントは，被制御および基底付の記憶域が割り振られたあと，それを解放するために用いられる。

制御ステートメント

一般に，PL／I プログラムの中のステートメントは，割込みが起ったり，次にあげる制御ステートメントのいずれかが実行されることによって，制御の流れが修正されない限り，順番に実行される。

```
GO TO
IF
DO
CALL
RETURN
END
STOP
```

GO TO ステートメント

GO TO ステートメントは，無条件ブランチとして使用される。GO TO の行先がラベルによって指定されているときは，値としていくつかのラベル定数を割り当てることによって，ラベル変数へのスイッチとして使用することができる。

ラベル変数に添字がついている場合は，添字を変えることによって，スイッチを制御することができる。GO TO ステートメントの行先は，ラベルの値を返す関数参照によっても指定することができる。ラベル変数または関数参照を使用することによって，非常に微妙なスイッチングの効果を得ることができる。ただし，制御ステートメントを簡単にする方が効率がよいことは，確かである。

GO TO ステートメントのキーワードは，1つまたは2つ以上のブランクで区切って2語にして書いてもよいし，GOTO と1語で書いてもよい。

IF ステートメント

IF ステートメントは，最もよく用いられる条件ブランチを設定するものであり，通常，IF という語の後に単純比較式といっしょに用いられる。たとえば，

```
IF A = B
```

```
THEN  真のときの処置
ELSE  偽のときの処置
```

THENまたはELSEクローズは，単一または複合ステートメント，DOグループ（「DOステートメント」の項を参照），または開始ブロックのいずれかから構成される。比較の結果が真ならば，THEN クローズが実行される。THEN クローズが実行されたあとは，ELSEクローズは実行されず，次のステートメントから実行が続けられる。THENクローズの中に，GO TO ステートメント，または制御を別のところに移すための何か他の制御ステートメントが含まれていてもよいことに注意していただきたい。

比較の結果が偽ならば，ELSE クローズだけが実行される。そのあと，制御は通常通り続けられる。

IF ステートメントは，次のように書くことができる。

```
IF A = B
   THEN C = D;
   ELSE C = E;
```

AがBと等しいときは，Dの値がCに割り当てられ，ELSEクローズは実行されない。AがBと等しくないときは，THEN クローズは実行されず，Eの値がCに割り当てられる。

THEN クローズまたはELSEクローズのどちらかに，ブランチ（条件付，無条件のどちらでも）を引き起すような制御ステートメントを含めることができる。たとえば，THENクローズにGO TOステートメントが含まれているとすれば，ELSE クローズを指定する必要はない。次の例について考えてみよう。

```
IF A = B
   THEN GO TO LABEL_1;
   次のステートメント
```

AがBに等しければ，THENクローズのGO TOステートメントによって，LABEL_1 に無条件にブランチする。AがBと等しくなければ，THEN クローズは実行されないので，次のステートメントがIFステートメントに関連するELSEクローズであるかどうかには関わりなく，制御は次のステートメントに渡される。

注：THENクローズによって制御の移動が引き起されず，しかもその後にELSEクローズもないという場合は，THEN クローズが実行されるかどうかには関係なく，次のステートメントが実行される。

IFというキーワードの後に書くことのできる式は，要素式だけである。配列式や構造式は許されない。しかし，2つ以上の演算子を使用する論理式は許される。たとえば，

```
IF A = B & C = D
   THEN GO TO R;
```

ネストされたIFステートメントを用いても，同じ種類のテストを行なうことができる。以下の3つの例は同じものである。

例1：

```
IF A = B & C = D
   THEN GO TO R;
B = B + 1;
```

例2：

```
IF A = B
   THEN IF C = D
           THEN GO TO R;
B = B + 1;
```

例3：

```
   IF A ¬= B THEN GO TO S;
   IF C ¬= D THEN GO TO S;
   GO TO R;
S:  B = B + 1;
```

DO ステートメント

DOステートメントは，ループを通るたびに制御変数を増やしながら指示された回数だけステートメントのグループを実行するよう指定するために，最もよく用いられる。このようなグループは，次のような形式にすることができる。

```
DO I = 1 TO 10;
   •
   •
   •
END;
```

反復して実行されるステートメントは，DOステートメントと関連するENDステートメントとによって区切らなければならない。上の場合は，制御変数Iの値が1から10まで変ってゆく間に，ステートメントのグループが10回実行される。このDOおよびENDステートメントの効果は，次のようにした場合と同じである。

```
    I = 1;
A:  IF I > 10 THEN GO TO B;
      •
      •
      •
    I = I + 1;
    GO TO A;
B:  次のステートメント
```

制御変数は，テストされる前に増やされること，および一般に，DOステートメントの中でセットされている限界を制御変数が越えたと

きにだけ，そのグループの後にあるステートメントに制御が移ることに注意していただきたい。最後の反復が終了したあとで制御変数を参照すると，変数の値は指定された限界よりも１だけ大きいものになっている。

DOステートメントは，制御変数を用いずに，WHILEオプションを用いて，次のように使用することもできる。

```
DO WHILE (A = B);
```

このステートメントをグループの先頭に置くと，Aの値がBの値と等しい限り，そのグループは繰返し実行される。

次の形式の制御変数とWHILEオプションを結びつけることができる。

```
DO I = 1 TO 10 WHILE (A = B);
```

このステートメントは，２つのテストを指定している。Ｉが増加するたびに，Ｉが１０を越えていないかどうかテストが行なわれる。そのあと，AがBに等しいかどうかをみる追加のテストが行なわれる。両方の条件が満足されたときにだけ，そのグループのステートメントが実行される。

１つのDOステートメントの中に，２つ以上の指定を含めることができる。次にあげるDOステートメントの各々について考えてみよう。

```
DO I = J,K;
DO I = 1 TO 10, 13 TO 15;
DO I = 1 TO 10, 11 WHILE (A = B);
```

最初のステートメントは，Ｉの値をＪの値と等しくセットして１回だけDOグループを実行し，さらにＩの値をKの値に等しくセットして１回だけ実行することを指定する。

２番目のステートメントは，合計１３回（Ｉの値を１から１０までにして１０回，さらにＩの値を１３から１５にして３回）DOグループを実行するよう指定している。３番目のDOステートメントは，最低１０回このグループを実行し，そのあと，（AとBが等しければ）さらに１回実行することを指定している。"１１"の後に"BY 0"を挿入した場合は，AとBが等しい限りＩが１１にセットされた状態で実行がつづけられる。どちらのステートメントの場合も，カンマを用いて２つの指定が区切られていることに注意していただきたい。これは，前の指定が満足させられたあとでだけ，後の指定を考慮することを表わしている。

DOステートメントの制御変数は，DOグループ内のステートメントの中で添字として使用することができるので，各々の反復でテーブルまたは配列の連続する要素を処理することができる。たとえば，

```
DO I = 1 TO 10;
   A(I) = I;
   END;
```

この例ではAの最初の１０個の要素は，それぞれ１，２，…，１０というようにセットされる。

反復指定の中の増分は，何か他の値が指定されていない限り（たとえば次のように），１とみなされる。

```
DO I = 2 TO 10 BY 2;
```

この例は，Ｉの値を２，４，６，８，および１０にしてループを５回実行することを指定している。

反復しないDOステートメント

DOステートメントで，DOグループのステートメントを繰返し実行することを，必ずしも指定しなくてもよい。単純なDOステートメントをDOグループと結びつけて，次のように用いることもできる。

```
DO;
 •
 •
 •
END;
```

このようにして単純なDOステートメントを使用している場合は，単に，DOグループを論理的に１つのステートメントとして扱うことを表わしている。この方法は，IFステートメントのTHENクローズまたはELSEクローズで実行されるいくつかのステートメントを指定するのに用いることができるが，このようにすれば，開始ブロックを用いなくても制御の順序を維持することができる。

CALL ,RETURN ,およびENDステートメント

サブルーチンは，そのサブルーチンの入口点を指名するCALLステートメントを用いて呼び出すことができる。サブルーチンの中でRETURN ステートメントが実行されたとき，またはENDステートメントが実行されてサブルーチンが終了したときに，活用化または呼出しをしたプロシージャーに制御が戻される。

関数参照に値を返すときは，括弧で囲んだ式をもつRETURNステートメントを関数プロシージャーの中で使用する。この形式は，関数参照によって呼び出されたプロシージャーから値を返すために用いられるものである。

プログラムが正常に終了するのは，主プロシージャーで最終的なENDステートメントが実行されるか，主プロシージャーでRETURN ステートメントが実行されて，呼出しプログラム（オペレーティング・システムであってもよい）に制御が戻ったときである。それ以外の方法によるプログラムの終了は，異常終了である。

STOPステートメント

STOP ステートメントは，プログラムを異常終了させるために用いられる。

例外制御ステートメント

これまでに説明してきた制御ステートメントは，実行されたときつねに制御の流れを変えるものである。実行の順序を変更することのできるもう一つの方法は，例外条件が起ったときにプログラム割込みを引き起す方法である。

一般的に云えば，例外条件とは，予期しないこと（たとえば桁あふれなど）が起こること，あるいは予期してはいるが起こる時点を予測できない（たとえば，ファイルの終りなど）ことが起こることである。このような条件の取扱いに関する詳細は，第14章「例外条件の取扱いとプログラム・チェックアウト」のところをみていただきたい。

例外制御ステートメントには，ONステートメント，REVERTステートメント，およびSIGNALステートメントの3つがある。

ONステートメント

ONステートメントは，この後で，指定された条件が起ったためにプログラム割込みが引き起されたときにとるべき処置を指定するために用いられるものである。ONステートメントは，様々な条件のいずれかに対してとるべき特定の処置を指定することができる。このような条件のいずれに対しても，PL／Iの一部として標準的なシステム処置が用意されており，割込みが起ったときにONステートメントが有効になっていなければ，標準的なシステム処置がとられる。ほとんどの場合，標準的なシステム処置は，メッセージを印刷して，実行を終らせることである。

ONステートメントは，次のような形式をとる。

ON 条件〔SNAP〕｛SYSTEM；｜オン・ユニット｝

"条件"には，H節「ON条件」のところにリストされている条件のいずれかを記入する。"オン・ユニット"は，その条件が生じて，割込みが起ったときにとるべき処置を指定する単一のステートメントまたは開始ブロックである。たとえば，

```
ON ENDFILE(DETAIL) GO TO NEXT_MASTER;
```

このステートメントは，DETAILという名前のファイルの終りを越えて読もうとして割込みが起ったときに，NEXT_MASTERというラベルのついたステートメントに制御を移すことを指定している。

オン・ユニットの実行がうまく完了したときは，割込みを引き起した条件如何によって，割込みのあった地点かまたはそのすぐ後の地点に，制御が正常に戻される。

ONステートメントの効果，すなわちオン・ユニットの設定は，(1)別のオン・ユニットまたはSYSTEMという語（標準のシステム処置に設定し直す）をもつ，同じ条件を指名する別のONステートメントを実行するか，または(2)その条件を指名するREVERTステートメントを実行することによって，1つのブロック内で変更することができる。別のブロックが活用化されるときに効力をもっていたオン・ユニットは，同じ条件に対する別のONステートメントが実行されない限り，活用化されたブロックおよびさらにそれによって活用化されたブロックの中でも効力をもち続ける。活用化を行なったブロックに制御が戻ったとき，オン・ユニットは元通りに再設定される。

REVERT ステートメント

REVERT ステートメントは，そのステートメントが記入されているブロックの中ですでに実行された，同じ条件に対するすべてのONステートメントの効力を取り消すために用いられるものである。

REVERT ステートメント（条件名を必ず指定しなければならない）は，現在実行中のブロックが呼び出されたときに，それを活用化したブロックの中で効力をもっていたオン・ユニットに設定し直す働きをもっている。

SIGNALステートメント

SIGNAL ステートメントは，指名された条件に対する割込みを起すよう促す働きをもっている。このステートメントは，ONステートメントを実行することによって設定されるオン・ユニットのコーディングをテストするために用いることができる。たとえば，

```
SIGNAL OVERFLOW;
```

このステートメントは，桁あふれによる割込みが起こるよう促し，OVERFLOW 条件に対して設定されたオン・ユニットを実行させるものである。オン・ユニットが設定されていない場合は，その条件に対する標準のシステム処置が実行される。

プリプロセッサー・ステートメント

PL／I では，コンパイレーション中に原始プログラムの内容をある程度まで制御できるようになっている。たとえば，プログラマーは，原始プログラムの中にある識別名を変更することを指定することができるし，プログラムの一部分だけを他の部分とは別にコンパイルするよう選択することも可能である。また，外部にある原始プログラムからテキストを組み入れることもできる。このような操作は，コンパイラーのプリプロセッサー段階で実行されるようになっており，原始プログラム自体の中にある他のステートメントの間に出てくるプリプロセッサー・ステートメントによって指定される。

一般に，プリプロセッサー・ステートメントは，キーワードの前にパーセント記号をつけることによって，識別される。そのうちのいくつかは，標準の PL／I ステートメントのキーワードと同じであり，コンパイル時におけるこの効果は，それが実行中に出てきたときと同じである。

次に，プリプロセッサー・ステートメントのすべてを記載したリストをあげる。

%ACTIVATE

%assignment

%DEACTIVATE

%DECLARE

%DO

%END

%GO TO

%IF

%INCLUDE

%ナル

%PROCEDURE

プリプロセッサー RETURN

これらのステートメントについては，第15章「コンパイル時の機能」とJ節「ステートメント」のところで説明してある。

リスティング制御ステートメント

印刷の際に行をスキップすることによって，プログラムのリスティングの形式をプログラマーが制御することができるようなステートメントが2つある。そのステートメントは次の2つである。

%PAGE

%SKIP

これらについては，第16章「コンパイル時の機能」のところをみていただきたい。

これらは先頭に%記号をつけてはいるが，プリプロセッサーを使用する必要はない。

第6章： プログラムの構成

この章では，PL／I プログラムを形成するときに，どのようにステートメントを構成してブロックにすることができるか，プログラムの中で，あるステートメントのブロックから別のブロックに制御がどのように流れてゆくか，ステートメントのブロックの中でデータに対してどのように記憶域を割り振ることができるか，ということについて説明する。

ブロック

ブロックは，区切られた一連のステートメントであり，プログラムのセクションを構成するものである。ブロックにすると，そのブロックの中で宣言された名前が局部的なものになり，変数の割振りが限定される。ブロックには，プロシージャー・ブロックと開始ブロックの2種類がある。このコンパイラーでは，1回のコンパイレーションで最高255個のブロックを受け入れることができる。

プロシージャー・ブロック

プロシージャー・ブロック（単にプロシージャーとも呼ばれる）は，以下のような形で，先頭にPROCEDUREステートメント，末尾にENDステートメントをつけて区切られた一連のステートメントである。

```
ラベル：〔ラベル：〕…  PROCEDURE;
                          ⋮
                        END〔ラベル〕;
```

プロシージャー名は，制御がプロシージャーに移るときにそこを通る主要な入口点であるから，すべてのプロシージャーに名前をつけなければならない。したがって，PROCEDUREステートメントには，少なくとも1個のラベルをつけなければならない。ENDステートメントのENDというキーワードの後にラベルを記入する必要はないが，もし記入するならば，そのENEステートメントに対応するPROCEDUREステートメントのラベル（またはいくつかのラベルのうちの1つ）と一致していなければならない。（例外がいくつかあるが，それらについては，この章の「ネストされたブロックおよびDOグループにおけるENDステートメントの用法」の項をみていただきたい。）次にあげるのは，プロシージャーの例である。

```
A:  READIN:  PROCEDURE;
             ステートメント－1
             ステートメント－2
               ⋮
             ステートメント－n
             END  READIN;
```

一般に，プロシージャーの名前（またはいくつかの名前のうちの1つ）を参照することによって，制御がプロシージャーに移される。したがって，前記の例のプロシージャーの場合は，その名前のいずれか（AまたはREADIN）を参照することによって，制御が与えられる。

PL／Iプログラムは，このようなプロシージャーが，1つまたは2つ以上集って構成されており，その各々に他のプロシージーおよび／または開始ブロックを含めることができる。

開始ブロック

開始ブロックは，以下のような形で，BEGINステートメントで始まり，ENDステートメントで終る1組のステートメントである。

```
〔ラベル：〕…  BEGIN;
                 ⋮
               END〔ラベル〕;
```

開始ブロックの場合は，プロシージャー・ブロックとは異なり，ラベルは任意選択である。BEGINステートメントの前に1つまたは2つ以上のラベルがついているときは，それは単にブロックの開始点を識別する働きしかしない。（そのブロックの名前を参照せずに，正規の順次的な実行を通じて，開始ブロックに制御を渡すことができる。ただし，GO TOステートメントを実行することによって，ラベル付のBEGINステートメントに制御を移すこともできる。）
ENDの後のラベルは，任意選択であるが，対応するBEGINステートメントのラベルと一致していれば，ENDの後にラベルを記入することができる。（例外がいくつかあるが，それらについては，この章の「ネストされたブロックおよびDOグループにおけるENDステートメントの用法」のところをみていただきたい。）次にあげるのは，開始ブロックの例である。

```
B:  CONTROL:  BEGIN;
              ステートメント－1
              ステートメント－2
                ⋮
              ステートメント－n
              END  B;
```

プロシージャーの場合とは異なり，開始ブロックの場合は，通常，それらを特別に参照することによって制御を渡すということはしない。ごく普通の場合のステートメント実行の際にとられる制御の順序が，そのまま開始ブロックの実行にも適用される。前のステートメントが実行されたあと，制御は順序通り開始ブロックに渡される。唯一の例外は，ONステートメントの中でオン・ユニットとして用

いられる開始ブロックの場合で，この場合は，指定された条件が起ったときにだけ，そのブロックが実行される。

開始ブロックは，PL／Iプログラムを構成する際に不可欠のものではない。しかし，プログラムのある特定の区域を区切るのに開始ブロックを使用すると便利な場合がある。このような利点については，本章および第7章「名前の認定」のところで説明する。

内部ブロックと外部ブロック

どのようなブロックも，1つまたは2つ以上のブロックを含むことができる。つまり，プロシージャーは，開始ブロックと同様，他のプロシージャーや開始ブロックを内蔵することができる。しかし，ブロックのオーバーラップは許されず，別のブロックを含むブロックは，そのブロックを完全に包含していなければならない。

別のブロックの中に含まれるプロシージャー・ブロックは，内部プロシージャーと呼ばれる。

別のブロックの中に含まれていないブロックは，外部プロシージャーと呼ばれる。

PL／Iプログラムには，少なくとも1つの外部プロシージャーがなければならない。（注：システム／360およびシステム／370のコンパイラーでは，各々の外部プロシージャーは別個にコンパイルされる。外部プロシージャーの入口名は，7桁を越えてはならない。）

開始ブロックはつねに内部ブロックであり，別のブロックに含まれていなければならない。

内部プロシージャーと開始ブロックは，ネストされたブロックとも呼ばれる。ネストされたブロックが，さらにまたその中にネストされたブロックを含んでいてもよい。一番外側のブロックは，つねにプロシージャーでなければならない。次の例について考えてみよう。

```
A： PROCEDURE；
    ステートメント－a1
    ステートメント－a2
    ステートメント－a3
    B： BEGIN；
        ステートメント－b1
        ステートメント－b2
        ステートメント－b3
        END B；
    ステートメント－a4
    ステートメント－a5
    C： PROCEDURE；
        ステートメント－c1
        ステートメント－c2
        D： BEGIN；
            ステートメント－d1
            ステートメント－d2
            ステートメント－d3
            E： PROCEDURE；
                ステートメント－e1
                ステートメント－e2
                END E；
            ステートメント－d4
            END D；
        END C；
    ステートメント－a6
    ステートメント－a7
    END A；
```

上の例で，プロシージャー・ブロックAは，他のどのブロックにも含まれていないので，外部プロシージャーである。ブロックBは，Aに含まれる開始ブロックであるが，その中に別のブロックは含んでいない。ブロックCは内部プロシージャーで，開始ブロックDを含んでおり，さらにDの中に内部プロシージャーEが含まれている。

この例では，Aから相対的に数えて，ネスティングのレベルは3つある。BとCが第1のレベル，Dが2番目のレベル（Cからみれば第1のレベル），Eが3番目のレベル（Cからは2番目，Dからは第1のレベル）である。許されるネスティングの最大の深度は，50である。

ENDステートメントの用法

プロシージャー，開始ブロック，またはDOグループでのENDステートメントの用い方には，次のような規則が適用される。

1. ENDの後にラベルがついていない場合は，そのENDステートメントよりも物理的に前にあってしかも最も近くにあるBEGINまたはPROCEDUREステートメントによって始まる。また閉じられていないブロック，またはDOステー

トメントによって始まる，まだ閉じられていないDOグループを閉じる（つまり終止させる）。

2. ENDの後に任意選択のラベルがついているときは，そのラベルと同じラベルをもち，しかも物理的にそのENDステートメントより前にあって最も近くにあるBEGIN, PROCEDURE，またはDOステートメントで始まる，まだ閉じられていないブロックまたはDOグループを閉じる。このようなブロックまたはDOグループの中にネストされている，まだ閉じられていないブロックまたはDOグループは，このENDステートメントによって自動的に閉じられる。この閉じ方を，複数終止という。

複数終止は，いくつものENDステートメントを連続して指定することにかわる簡便な方法である。実際には，コンパイラーは，複数終止を指定するENDステートメントの直前に，必要な数のENDステートメントを挿入する。たとえば，次のような外部プロシージャーが定義されているものとしよう。

```
FRST:  PROCEDURE;
          ステートメント −f1
          ステートメント −f2
          ABLK:  BEGIN;
                    ステートメント −a1
                    ステートメント −a2
                    SCND:  PROCEDURE;
                              ステートメント −s1
                              BBLK:  BEGIN;
                                        ステートメント −b1
                                        END;
                              END;
                    ステートメント −a3
                    END ABLK;
          END FRST;
```

この例では，開始ブロックBBLKと内部プロシージャーSCNDは実質的に同じ場所で終了する。つまり，各々のENDステートメントの間には，何もステートメントがない。このことは，開始ブロックABLKと外部プロシージャーFRSTの場合も同様である。このような場合は，上例のように各ブロックに対してENDステートメントを使用する必要はなく，BBLKとSCNDを終了させるために1つのENDステートメント，またABLKとFRSTを終了させるためにもう1つのENDステートメントを使用すればよい。

前者の場合は，ラベルをつけずにENDステートメントを使用すると，開始ブロックBBLKしか閉じられない（前記の最初の規則を参照）ので，END SCNDというステートメントを用いる。

後者の場合は，END FRSTだけで充分である（END ABLKというステートメントは余分である）。したがって，上記の例は，次のように書きかえることができる。

```
FRST:  PROCEDURE;
          ステートメント −f1
          ステートメント −f2
          ABLK:  BEGIN;
                    ステートメント −a1
                    ステートメント −a2
                    SCND:  PROCEDURE;
                              ステートメント −s1
                              ステートメント −s2
                              BBLK:  BEGIN;
                                        ステートメント−b1
                                        ステートメント−b2
                              END SCND;
                    ステートメント −a3
          END FRST;
```

複数終止を指定するENDステートメントにつけられたラベル接頭語は，最後のENDステートメントに適用されるものとみなされる。したがって，このようなステートメントに制御が渡されると，その間にあるすべてのグループとブロックは，終了する。たとえば，

```
CBLK:     PROCEDURE;
          ステートメント −c1
          ステートメント −c2
DGP:      DO I = 1 TO 10;
          ステートメント −d1
          GO TO LBL;
          ステートメント −d2
LBL:      END CBLK;
```

上の例では，END CBLKというステートメントは，CBLKというブロックと反復DOグループDGPを閉じる。その結果，DGPに対するラベルなしのENDステートメントがステートメント−d2の直後にあるのと同じになるので，LBLに制御が移ると，DGPの反復は最初の回をのぞいてすべて阻止される結果になり，ステートメント −d2は実行されないことになる。

ブロックの活用化

開始ブロックとプロシージャーは，物理的には類似しており，また記憶域の割振りや解放，名前の有効範囲の限定などの点でも同じような働きをするが，その活用化や実行の方法の点で，互いに異なっている。

開始ブロックは，単一のステートメントと同じように，プログラムの正規の流れの順序の課程で活用化されて実行され（オン・ユニットとして指定されたときは例外），一般に，単一のステートメントを書けるところであればどこにでも置くことができる。

それに対してプロシージャーの場合は，正規のプログラムの流れは，プロシージャーを避けて，PROCEDUREステートメントの前のステートメントから，そのプロシージャーのENDステートメントの後にあるステートメントに制御が渡される。プロシージャーを活用化することができるのは，プロシージャー参照だけである。

プロシージャー参照とは，次の文脈のいずれかの中に，入口式として書かれるものである。

1. CALLステートメントのCALLというキーワードの後

2. INITIAL属性のCALLオプションの中のCALL というキーワードの後

3. 関数参照として

この章では，上記のうち最初のものの例を用いるが，その題材は，他の2つの形式にも関連している。さらにくわしい点については，I節「属性」のINITIAL属性のところ，および9章「サブルーチンと関数」をみていただきたい。

CALLステートメントの最も簡単な形式は，以下の通りである。

```
CALL  入口定数；
```

入口定数がPROCEDUREステートメントのラベルであるときは，それは，プロシージャーへの主要入口点を表わしている。ENTRYステートメントのラベルであるときは，副次入口点を表わしている。次にあげるのは副次入口点をもつプロシージャーの例である。

```
             A：  PROCEDURE；
                  ステートメント－1
                  ステートメント－2
          ERRT：  ENTRY；
                  ステートメント－3
                  ステートメント－4
                  ステートメント－5
  NEXT：  RETR：  ENTRY；
                  ステートメント－6
                  ステートメント－7
                  ステートメント－8
                  END  A；
```

この例では，Aがプロシージャーへの主要入口点であり，ERRT，NEXT，およびRETRは副次入口点を指定するものである。実際には，NEXTとRETRは同じENTRYステートメントのラベルであるから，これらは両方とも同じ副次入口点を指定している。このプロシージャーは，以下のステートメントのいずれかによって活用化することができる。

```
CALL  A；
CALL  ERRT；
CALL  NEXT；
CALL  RETR；
```

かわりに，適当な入口名の値を入口変数に割り当てて，この入口変数をプロシージャー参照の中で使用することができる。以下の例では，2つのCALLステートメントは同じ効果をもっている。

```
DECLARE  ENT1  ENTRY  VARIABLE；
      .
      .
      .
ENT1  =  ERRT；
      .
      .
      .
CALL  ENT1；
      .
      .
      .
CALL  ERRT；
```

プロシージャー参照が実行されると，指定された入口点を含むプロシージャーが活用化されるが，このことは，呼び出されるともいう。その結果，制御は指定された入口点に移る。プロシージャー参照が行なわれている地点は，呼出し点と呼ばれ，参照が行なわれているブロックは，呼出し側ブロックと呼ばれる。呼出し側ブロックは，それが呼び出したブロックに制御が移っても，活用化されたままである。

主要入口点でプロシージャーが呼び出されたときはつねに，呼び出されたプロシージャーの中で最初に実行可能なステートメントから実行が始まる。

しかし，副次入口点でプロシージャーが呼び出されたときは，その副次入口点を定義しているENTRY ステートメントの次にある。最初の実行可能なステートメントから実行が始まる。

したがって，最後にあげた例の中で番号のつけられたステートメントがすべて実行可能であるとすれば，CALL A というステートメントは，その主要入口点でプロシージャーAを呼び出し，ステートメント－1から実行が始まる。CALL ERRTというステートメントは，副次入口点 ERRT でプロシージャーAを呼び出すので，ステートメント－3から実行が始められる。CALL NEXTまたはCALL RETRというステートメントはいずれも，もう1つの副次入口点でプロシージャーAを呼び出すので，ステートメント－6から実行が始められる。

順次的な流れの最中にENTRY ステートメントに出合っても，それは実行されないことに注意していただきたい。制御はENTRYステートメントを，注釈とみなして飛び越してゆく。

どのようなプロシージャーも（外部，内部のいずれも），外部プロシージャーはつねに呼び出すことができるが，何か他のプロシージャーに含まれている内部プロシージャーをつねに呼び出すことができるとは限らない。外側のプロシージャーからみて第1のレベルでネスティングされている内部プロシージャーは，つねにその外側のプロシージャーから，あるいは互いに呼び出し合うことができる。たとえば，

```
PRMAIN:  PROCEDURE;
         ステートメント－1
         ステートメント－2
         ステートメント－3
         A:  PROCEDURE;
             ステートメント－a1
             ステートメント－a2
             B:  PROCEDURE;
                 ステートメント－b1
                 ステートメント－b2
             END A;
         ステートメント－4
         ステートメント－5
         C:  PROCEDURE;
             ステートメント－c1
             ステートメント－c2
             END C;
         ステートメント－6
         ステートメント－7
         END PRMAIN;
```

この例では，PRMAINはプロシージャーAおよびCを呼び出すことができるが，Bを呼び出すことはできない。プロシージャーAはプロシージャーBおよびCを呼び出すことができる。プロシージャーBはプロシージャーCを呼び出すことができる。また，プロシージャーCはプロシージャーAを呼び出すことはできるが，Bを呼び出すことはできない。

ブロックの活用化に関するこれまでの説明は，プログラムがすでに活用化されていることを前提としたものであった。PL/Iプログラムは，呼出しプログラムが第1プロシージャーを呼び出したときに活用化される。この呼出しプログラムは通常，オペレーティング・システムである(他のプログラムであってもよいが)。第1プロシージャー(主プロシージャーともいう)は，外部プロシージャーでなければならず，次の例で示すように，そのPROCEDUREステートメントは，OPTIONS(MAIN)の指定をもつものでなければならない。

```
CONTRL:  PROCEDURE OPTIONS(MAIN);
         CALL A;
         CALL B;
         CALL C;
         END CONTRL;
```

この例では，CONTRLが第1プロシージャーであり，プログラム内の他のプロシージャーを呼び出している。

ブロックの活用化についてすでに述べたこと，あるいは暗示したことを，以下に要約する。

- オペレーティング・システムによって第1プロシージャーが活用化されたときに，プログラムは活用化される。
- 第1プロシージャーを除いて，プログラムの中に含まれている外部および内部プロシージャーは，プロシージャー参照によって呼び出されたときにだけ活用化される。
- 開始ブロックは，正規の順次的な流れを通して，あるいはオン・ユニットとして活用化される。
- 第1プロシージャーは，そのプログラムの間中，活用化されたままである。
- 活用化されたブロックはすべて，それらが終了させられるまで(以下参照)，活用化の状態にある。

ブロックの終了

通常，プロシージャー・ブロックは，プロシージャー参照以外の方法で，呼出し側ブロックまたは活動中の他のブロックに制御が移ったときに，終了する。同様に，開始ブロックは，プロシージャー参照以外の方法で，別の活動中のブロックに制御が移ったときに，終了する。このような制御の移動の方法は幾通りもあるが，その解釈は，終了するブロックのタイプによって異なっている。

開始ブロックの終了

開始ブロックは，次のいずれかが起ったときに終了する。

1. そのブロックに対するENDステートメントに制御が達したとき。このような場合は，そのブロックがオン・ユニットであるときを除いて，物理的にENDの次にあるステートメントに制御が移る。
2. 開始ブロック(またはその開始ブロックの中で活用化されたブロック)の中でGO TOステートメントが実行された結果，そのブロックの中に含まれていない地点に制御が移ったとき。
3. STOPステートメントが実行された(それによって実行が停止した)とき。
4. 開始ブロックの外側およびそれが含んでいるプロシージャーの外側に制御を移すRETURNステートメントに，制御が達したとき。

項目のところで述べたタイプのGO TOステートメントは，次のように，他のブロックを終了させることもある。

移動の地点が，終了させられたブロックを直接活用化したのではないブロックに含まれているときは，活用化の順序の中に介在しているブロックはすべて，終了する。

たとえば，開始ブロックBが開始ブロックAに含まれているとすれば，AにもBにも含まれていない地点に制御を渡すGO TOステートメントがBの中で与えられると，AとBは両方とも実質的に終了させられる。このような例が，次に示してある。

```
FRST:  PROCEDURE OPTIONS(MAIN);
       ステートメント－1
       ステートメント－2
       ステートメント－3
       A:  BEGIN;
           ステートメント－a1
           ステートメント－a2
           B:  BEGIN;
               ステートメント－b1
               ステートメント－b2
               GO TO LAB;
               ステートメント－b3
               END B;
           ステートメント－a3
           END A;
       ステートメント－4
       ステートメント－5
 LAB:  ステートメント－6
       ステートメント－7
       END FRST;
```

FRSTが呼び出されたあと，最初の3つのステートメントが実行され，次に開始ブロックAが活用化される。Aの中にある最初の2つのステートメントが実行され，次に開始ブロックBが活用化される（Aは活用化されたままである）。Bの中のGO TOステートメントが実行されると，FRSTの中のステートメント－6に制御が移る。ステートメント－6はAにもBにも含まれていないので，AとBは両方とも終了する。このように，開始ブロックBの外に制御が移ると，ブロックBが終了するとともに，間に介在するブロックAも終了する結果になる。

プロシージャーの終了

次のいずれかが起こると，プロシージャーが終了する。

1. そのプロシージャーの中にあるRETURN ステートメントに制御が達したとき，RETURN ステートメントが実行されると，呼出し側プロシージャー内の呼出し地点に制御が戻る。呼出し地点がCALLステートメントの場合は，呼出し側プロシージャーの実行は，CALLの次にあるステートメントから再開される。呼出し地点が，他の形式のプロシージャー参照（CALLオプションまたは関数参照）である場合は，その参照を含んでいるステートメントの実行が再開される。

2. そのプロシージャーのENDステートメントに制御が達したとき。これは，RETURN ステートメントを実行したのと，実質的に同じになる。

3. そのプロシージャー（またはそのプロシージャーの中から活用化されたブロック）の中でGO TOステートメントが実行されて，そのプロシージャーの中に含まれていない地点に制御が移ったとき。

4. STOP ステートメントが実行された（その結果，実行が停止させられた）とき。

項目1と2は，プロシージャーの正常な終了である。項目3と4は，異常終了である。

開始ブロックの場合と同様，項目3で述べられているタイプの停止があると，いくつかのプロシージャーおよび／または開始ブロックが終了してしまうこともある。とくに，GO TOステートメントによって指定された移動の地点が，終了させられたブロックを直接活用化したのではないブロックに含まれているときは，活用化の順序の中に介在するすべてのブロックが終了する。次の例について考えてみよう。

```
  A:  PROCEDURE OPTIONS(MAIN);
      ステートメント－1
      ステートメント－2
      B:  BEGIN;
          ステートメント－b1
          ステートメント－b2
          CALL C;
          ステートメント－b3
          END B;
      ステートメント－3
      ステートメント－4
      C:  PROCEDURE;
          ステートメント－c1
          ステートメント－c2
          ステートメント－c3
          D:  BEGIN;
              ステートメント－d1
              ステートメント－d2
              GO TO LAB;
              ステートメント－d3
              END D;
          ステートメント－c4
          END C;
      ステートメント－5
LAB:  ステートメント－6
      ステートメント－7
      END A;
```

上の例では，AがBを活用化し，そのBがCを活用化し，さらにそのCがDを活用化する。Dの中で，GO TO LABというステートメントによって，Aの中のステートメント－6に制御が渡される。このステートメントは，D，C，またはBのいずれにも含まれていないので，この3つのブロックはすべて終了する。Aは活用化の状態のままである。このように，Dの外に制御が移ると，ブロックDが終了するとともに，間に介在するブロックBとCも終了することになる。

プログラムの終了

次のいずれかが起ったときは，プログラムが終了する。

1. プログラムの制御がSTOP ステートメントに達したとき，これは異常終了である。

2. 主プロシージャーの中のRETURNステートメントまたは最後のENDステートメントに制御が達したとき。これは正常な終了である。

3. ERROR条件に対するオン・ユニットが実行されて，正常に戻ったとき（つまり，GO TOステートメントによって，オン・ユニットの外に制御が移らなかったとき）。

プログラムが終了するときは正常な場合も異常な場合も，呼出しプログラム（通常は，オペレーティング・システムの制御プログラム）に制御が戻る。

記憶域の割振り

記憶域の割振りは，変数によって表わされるデータ項目を内部的に記録することができるように，記憶域と変数とを結びつけるための処理である。変数に記憶域が結びつけられたとき，その変数は割り振られたという。ある変数に対する割振りは，静的に（つまり，プログラムの実行の前に）行なうことも，また，実行中に動的に行なうこともできる。静的に割り振られる変数は，プログラムの間中，割り振られたままになっている。動的に割り振られる変数は，記憶域のクラスに応じて，その変数が含まれているブロックが終了したときか，またはプログラマーが要求したときにその記憶域を放棄する。

変数に対して記憶域が割り振られる方式は，その変数の記憶域クラスによって決まる。記憶域クラスには，静的，自動，被制御，および基底付の4つがある。各々の記憶域クラスは，それぞれ，該当する記憶域クラスの属性STATIC, AUTOMATIC, CONTROLLEDおよびBASEDを用いて指定する。後の3つは，動的な記憶域割振りを定義するものである。

要素，配列，および大構造の各変数に対して，記憶域クラス属性を明示的に宣言することができる。変数が配列，または大構造変数である場合は，その変数に対して宣言された記憶域クラスが，配列または構造の中のすべての要素に適用される。

記憶域クラスの属性について明示的に宣言されていない変数にはすべて，AUTOMATIC属性が与えられる。ただし，例外が1つあり，EXTERNAL属性をもつ変数には，STATIC属性が与えられる。

第8章で，各種の記憶域クラスの用法について説明する。

活用状態にあるプロシージャーの再活用化（回帰）

すでに活用状態にあるプロシージャーのうち，自分自身または活用状態にある別のプロシージャーから再び活用化することができるものを，回帰的プロシージャーという。このような再活用化は回帰と呼ばれる。

PROCEDUREステートメントの中でRECURSIVEオプションが指定されているときにだけ，そのプロシージャーを回帰的に呼び出すことができる。このオプションは，プロシージャーが副次入口点をもっていれば，その名前にも適用される。

回帰的プロシージャーが呼び出されるたびに，その環境記録（自動変数の値，など）がとっておかれる。つまり，この環境は，回帰的呼出しのときに押し下げられ，その呼出しが終了したときに浮かび上がるものと考えることができる。ラベル定数には，そのラベルが入っているブロックの現在の呼出しを識別する情報がつねに入っていることに注意していただきたい。

```
RECURS: PROCEDURE RECURSIVE;
   DECLARE X STATIC EXTERNAL INITIAL(0);
     ⋮
   X = X + 1;
   PUT DATA (X);
   IF X = 5 THEN GO TO LAB;
   CALL AGN;
   X = X - 1;
   PUT DATA (X);
     ⋮
   LAB: END RECURS;
AGN: PROCEDURE RECURSIVE;
   DECLARE X STATIC EXTERNAL INITIAL(0);
     ⋮
   X = X + 1;
   PUT DATA (X);
     ⋮
   CALL RECURS;
   X = X - 1;
   PUT DATA (X);
   END AGN;
```

上の例の，RECURSとAGNは両方とも回帰的プロシージャーである。Xは静的変数であり，INITIAL属性をもっているので，プログラムの実行が始まる前に割振りが行なわれ，初期値化される。

最初にRECURS が呼び出されるとき，Xは1だけ増やされ，PUTステートメントによってX＝1が転送される。Xが5より小さいので，AGNが呼び出される。AGNの中で，Xが1だけ増やされ，X＝2が転送される（やはり，PUTステートメントによって）。そのあと，AGNはRECURS を再び呼び出す。

2度目のRECURS の呼出しは，RECURS がまだ活用状態にあるので，回帰的呼出しである。前と同じようにXが増やされ，そのあと，X＝3が転送される。Xはまだ5より小さいので，AGNが再び呼び出される。この呼出しのときAGNは活用化されているので，AGNのこの呼出しも回帰的である。Xがまた増やされ，X＝4が転送され，さらにRECURS の3回目の呼出しが行なわれる。

RECURS の3回目の呼出しでは，X＝5が転送される。しかし，Xがもはや5より小さくなくなったので，GO TO LABが実行され，RECURS は終了する。しかし，これは，RECURSの3度目の呼出しだけが終了したのであって，制御は，RECURSを3度目に呼び出したプロシージャーに戻ることになる。つまり，AGNの2度目の呼出しの中にあるCALL RECURSの後にあるステートメントに制御が戻る。この地点で，Xが1だけ減らされ，X＝4が転送される。そのあと，AGNの2回目の呼出しが終了し，2回目にAGNを呼び出したプロシージャーに制御が戻る。つまり，RECURS の2回目の呼出しの中にあるCALL AGNの後のステートメントに制御が戻るわけである。ここで，Xはまた減らされ，X＝3が転送され，そのあと，RECURS の2回目の呼出しが終了して，AGNの最初の呼出しに制御が戻る。Xが再び減らされ，X＝2が転送され，AGNの最初の呼出しが終了し，RECURSの最初の呼出しに制御が戻る。Xが減らされて，X＝1が転送され，RECURS の最初の呼出しが終了する。このあと，最初にRECURS を呼び出したプロシージャーに制御が戻る。

特定の呼出しの中でラベル変数にラベル定数が割り当てられる場合は，別の呼出しの中でその変数を指名するGO TOステートメントを用いれば，割当てが行なわれたときの環境を復元することができることに注意していただきたい。

プロローグとエピローグ

ブロックを活用化するたびに，行なわなければならないある種の作業があり，このあとで始めて，そのブロック内の最初の実行可能なステートメントに制御が達することができる。このような一連の作業を<u>プロローグ</u>という。同様に，ブロックが終了するときも，制御をそのブロックの外に出すためには，ある種の作業を行なわなければならない。このような一連の作業を<u>エピローグ</u>という。

プロローグとエピローグは，コンパイラーが行なうべき仕事であり，プログラマーの責任ではない。しかし，これらについて知っておくと，プログラムの性能を上げる上でプログラマーの役に立つこともあるので，ここで説明する。

プロローグ

プロローグは，ブロックの活用化の最初の段階で実行される，コンパイルされたコードである。一般に，プロローグによって行なわれる作業は，次のようなものである。

- 自動変数およびDEFINED 変数の次元の限界とストリングの長さを計算する。
- 自動変数の記憶域の割振りと初期値化（指定されていれば）を行なう。
- 現在活用化されているブロックのうち，どれがこのプロシージャーに知られているかを調べて，自動記憶域に関して正しい世代のものをアクセスすることができるように，また，正しいオン・ユニットに入ることができるようにする。
- このブロックから見かけ引数が渡されるときは，それに対して記憶域を割り振る。

プロローグは，式で表わされた初期値（反復因数も含む）や，配列の限界，ストリングの長さ，エリアの大きさなどを評価しなければならないこともある。

プログラムの中にある各ブロックに対して，コンパイラーは，次のような順序でこれらの値を割り当てる。

1. ブロック内の他の宣言とは無関係の値　（外側のブロックからその値がとられることもある。）
2. ブロック内の他の宣言によって決まる値。ブロック内の2つ以上の他の宣言によって値が決まる場合は，正しく初期値化できるかどうか保証できない。たとえば，次の例では，

```
DCL I INIT(10), J INIT(I), K INIT(J);
```

Kの初期値化は，正しくできるかどうか保証できない。

データ項目の宣言は，互いに依存し合うものであってはならないことに注意していただきたい。たとえば，次のような宣言は誤りである。

```
DCL A(B(1)), B(A(1));
```

互いに依存し合うという関係は，3つ以上のデータ項目の間で起こり得ることに注意していただきたい。たとえば，次のような宣言も誤りである。

```
DCL A(B(1)), B(C(1)), C(A(1));
```

エピローグ

エピローグは，ブロックの終了の最終段階で実行される，コンパイルされたコードである。一般に，エピローグによって行なわれる作業には，次のようなものがある。

- そのブロックが活用化される前に存在していたオン・ユニットの環境に設定し直す。
- そのブロックの中で割り振られたすべての自動変数に対する記憶域を解放する。

第7章： 名前の認定

PL/Iプログラムは，識別名，定数，および演算子や区切り符号として用いられる特殊文字から構成される。識別名自体は，キーワードであっても，プログラマーが指定した意味をもつ名前であってもよい。PL/I言語は，ある識別名がキーワードであるかどうかコンパイラーが文脈から判断できるように作られているので，プログラマーが定義する名前として使用できないような予約語のリストは存在しない。どのような識別名も，名前として使用することができる。この場合の制約は，プログラム内のある任意の地点では，ある1つの名前は1つの意味しかもっていてはならない，ということだけである。たとえば，ファイル変数と浮動小数点変数の両方に，同じ名前を使用してはならない，ということである。

注：上で述べたことは，60文字セットを使用している限り正しいが，48文字セットを用いて書かれるプログラムの場合は，プログラマーが定義する識別名として使用できない識別名がいくつかある。このような識別名は，GT，GE，NE，LT，NG，LE，NL，CAT，OR，AND，NOT，およびPTである。

しかし，ある名前が，プログラム全体を通じて同じ意味をもっている必要はない。あるブロックの中で宣言された名前は，そのブロックの中でだけ1つの意味をもっている。同じ名前が外側のブロックでも宣言されていない限り，ブロックの外側ではその名前は知られない。このような場合，外側のブロックにある名前は，別のものを指している。このようなことから，プログラマーは，局部的な定義をすることが可能であり，したがって，プログラムの別の部分を書いている他のプログラマーが使用している名前をすべて知っていなくても，プロシージャーや開始ブロックを書くことができる。

1つの名前が2つ以上の意味をもつこともできるので，ある特定の意味は，プログラムのどの部分に適用されるか定義することが重要な問題になってくる。PL/Iでは，宣言によって（必ずしも明示的である必要はない），名前に属性と意味が与えられる。その意味が適用されるプログラムの部分を，その名前の宣言の有効範囲という。多くの場合，名前の有効範囲は，プログラムの中でその名前が宣言された位置（または，明示宣言されない場合は宣言されたとみなされる位置）によって，完全に決まる。同じ名前をもつデータに関して，2つ以上の世代がある場合（たとえば，回帰の場合）もあるが，このような場合は，別々に考慮される。

名前の有効範囲に関する規則を理解するためには，「……に含まれている」という語と「……の内部にある」という語について理解する必要がある。

……に含まれている：

PROCEDUREまたはBEGINステートメントから該当するENDステートメントまでの間にある。ブロックのテキストはすべて，そのブロックに含まれているという。ただし，そのブロックの先頭にあるBEGINまたはPROCEDUREステートメントのラベルと，そのブロックに適用されるENTRYステートメントのラベルは，そのブロックの中には含まれないことに注意していただきたい。ネストされているブロックは，それが入っているブロックに含まれている。

……の内部にある：

ブロックに含まれてはいるが，その中にネストされている他のブロックには含まれていないテキストは，そのブロックの内部にあるという。プロシージャーの入口名（およびBEGINステートメントのラベル）は，そのブロックに含まれないことに注意していただきたい。したがって，これらはそれを含んでいるブロックの内部にあるわけである。外部プロシージャーの入口名は，外部プロシージャーの外側にあるものとして取り扱われる。

これらの用語のほかに，いろいろなタイプの宣言も重要である。宣言のタイプには，明示宣言，文脈宣言，および暗示宣言の3つがあるが，これらについては，次のセクションで説明する。

明示宣言

名前が次のものの中にあった場合は，明示宣言されたことになる。

1. DECLAREステートメントの中
2. パラメータ・リストの中
3. ステートメント・ラベルとして
4. PROCEDUREまたはENTRYステートメントのラベルとして

パラメーター・リストの中に名前があるということは，その名前に対するDECLAREステートメントが，パラメーター・リストが入っているPROCEDUREまたはENTRYステートメントの直後にあるのと同じである（ただし，同じブロックの内部にあるDECLAREステートメントの中で同じ名前を宣言してもよい）。

PROCEDUREまたはENTRYステートメントのラベルとして名前が記入されているということは，それが参照しているプロシージャーに対するPROCEDUREステートメントの直前に，DECLAREステートメントを置き，その中で宣言したのと同じである。

ステートメントにラベル接頭語をつけることは，ラベルを明示宣言するのと同じである。

明示宣言の有効範囲

ある名前の明示宣言の有効範囲は，その中に含まれているすべてのブロックも含めて，その宣言がその内部にあるブロックである。ただし，同じ識別名に関する別の明示宣言がその内部にあるようなブロック（およびその中に含まれているブロック）はのぞく。

たとえば，

```
                                   P A B B'C C'D Q R
P: PROCEDURE;
   DECLARE A, B;
   Q: PROCEDURE;
      DECLARE B, C;
      R: PROCEDURE;
         DECLARE C, D;
         END R;
      END Q;
   END P;
```

右側の線は，名前の有効範囲を示している。BとB'は，Bという名前の2通りの用法を表わしており，CとC'は，Cという名前の2通りの用法を表わしている。

文脈宣言

名前が特定の文脈にあらわれたときは，その属性のいくつかは明示宣言をしなくても，判定することができる。このような場合に，名前のあらわれた位置が，同じ名前に対する明示宣言の有効範囲内でなければ，その名前は，文脈的に宣言されたという。

明示的に宣言されなかった名前は，次のような場合に文脈的に認定され，宣言される。

1. CALLステートメントまたはCALLオプションの中にある名前，または引数リストが後にある名前には，BUILTINおよびINTERNAL属性が与えられる。のちにその名前がPL/Iの組込み関数またはサブルーチンの名前でないことがわかった場合には，その属性はENTRYとEXTERNALに変えられる。

   ただし，PL/Iチェックアウト・コンパイラーの場合は，PL/Iの組込み関数名に似た名前は，スペルの誤りとみなされて，「訂正される」ことに注意していただきたい。引数をもたない組込み関数と擬似変数（ONCHARとONSOURCEなど）は，BUILTIN属性をつけて明示的に宣言するか，ナル引数リストを用いて文脈的に宣言するか（たとえばONCHAR（）という形で），あるいはDEFAULTステートメントを用いて，たとえば次の例のように，暗示的に宣言しなければならない。

   ```
   DEFAULT RANGE(ON, DATE) BUILTIN;
   ```

2. FILEまたはCOPYオプションの中にある名前，またはファイル名を必要とする条件に対するON，SIGNAL，またはREVERTステートメントの中にある名前には，FILE属性が与えられる。

3. ON CONDITION，SIGNAL CONDITION，またはREVERT CONDITIONステートメントの中にある名前は，プログラマーが定義した条件名として認定される。

4. EVENTオプションまたはWAITステートメントの中にある名前には，EVENT属性が与えられる。

5. BASED属性の中，SETオプションの中，またはロケーター修飾記号の左側にある名前には，POINTER属性が与えられる。

6. INオプション，またはOFFSET属性の中にある名前には，AREA属性が与えられる。

次にあげるのは，文脈宣言の例である。

```
READ FILE (PREQ) INTO (Q);
ALLOCATE X IN (S);
```

上記のステートメントでは，PREQにはFILE属性が与えられ，SにはAREA属性が与えられる。

文脈宣言の有効範囲

文脈宣言の有効範囲は，その名前が出てきた外部プロシージャーのPROCEDUREステートメントの直後にDECLAREステートメントを置き，その中で宣言された場合と同じように決められる。

文脈宣言は，その名前が外部プロシージャーの中で宣言された場合と同じ効果をもつことに注意しなければならない。これは，その文脈宣言のもとになったステートメントが，外部プロシージャーの中に入っているブロック（たとえば，Bという名前とする）の内部にある場合であっても同様である。

したがって，この名前は，同じ名前が明示的に宣言されているブロックをのぞいて，外部プロシージャー全体にわたって知られることになる。これは，ブロックBを含んでいる外部プロシージャーからその宣言を，ブロックBが受けついだのと同じである。

明示宣言の有効範囲内で，文脈宣言が行なわれることはあり得ないので，明示宣言によってその名前に与えられた属性に，文脈によって追加をすることはできない。たとえば，次のプロシージャーは誤りである。

```
P:  PROC (F);
      :
    READ FILE(F) INTO(X);
      :
    END P;
```

識別名Fは，パラメーター・リストの中にあるので，明示宣言されたことになる。これに対する標準の省略時解釈の属性はREAL DECIMAL FLOATであり，FILEオプションの中にある場合にFに対して通常与えられる属性とは矛盾する。このような識別名の使い方は，誤りである。

暗示宣言

プログラムの中に出てくる名前が，明示的にも文脈的にも宣言されていない場合は，この名前は暗示宣言されているという。暗示宣言の有効範囲は，その名前が使用されている外部プロシージャーのPROCEDUREステートメントのすぐ後で，DECLAREステートメントによって宣言されたのと同じようにして，決められる。中に含まれているプロシージャーの中でだけ用いられる名前は，外側のプロシージャーの中で知られることになる。

DEFAULTステートメントを用いて，標準の省略時解釈が，プログラマーが定義した省略時解釈に置き換えられていない限り，暗示宣言が行なわれると，名前の頭文字にしたがって，標準の省略時解釈の属性が適用される。名前がIからNのいずれかの文字で始まっている場合は，REAL FIXED BINARY (15,0)という属性が与えられる。それ以外の文字（英文の拡張である$，#，@も含めて）で始まる名前には，REAL FLOAT DECIMAL (6) という属性が与えられる。

各種宣言の例

データの宣言の有効範囲が図7.1に示されている。左側の大括弧はブロックの構造を示しており，右側の大括弧はそれぞれの名前の宣言の有効範囲を示している。この図の中で，QとRに関する2通りの宣言の有効範囲は，QとQ'，RとR'というように表わしてある。

Pは，ブロックAで宣言されており，しかも再宣言されていないので，A全体に知られる。

Qは，Aで宣言され，Bで再宣言されている。最初の宣言の有効範囲はBをのぞいたA全体であり，2番目の宣言の有効範囲はブロックBだけである。

Rは，ブロックCで宣言されているが，ブロックBの中でもRへの参照が行なわれている。ブロックBの中でRを参照すると，外部プロシージャーであるAの中でRを暗示宣言したことになる。したがって，有効範囲の異なる2つの別の名前が存在することになる。明示宣言されたRの有効範囲はCであり，暗示宣言されたRの有効範囲は，ブロックCを<u>のぞく</u>A全体である。

Iは，ブロックCの中で参照されているので，外部プロシージャーAの中で暗示宣言が行なわれることになる。その結果，この宣言は，A全体（その中に含まれているプロシージャーB，C，Dも含めて）に適用される。

Sは，プロシージャーDの中で明示宣言されており，Dの中でだけ知られている。

入口定数およびステートメント・ラベルの宣言の有効範囲が，図7.2に示してある。この例では，2つの外部プロシージャーが示されている。これらのプロシージャーAおよびEの名前は，それが適用されるプロシージャーの中でEXTERNAL 属性をつけて明示宣言されているものとみなされる。さらに，EはAの中でも外部入口定数として明示宣言されている。Eの明示宣言は，ブロックA全体に適用されるものであって，ブロックE全体に適用されるEの明示宣言には連結されない。名前Eの有効範囲は，ブロックA全体とブロックE全体である。名前Aの有効範囲は，ブロックA全体だけであり，Eは有効範囲ではない。しかし，これは，Eの中で外部入口宣言に記入することも可能であり，このようにすると，Aの有効範囲は，A全体とE全体になる。

ラベルL1は，Aの内部にあるステートメントとCの内部にあるステートメントのところに出て来ている。したがって，別々の2つの宣言が設定される。最初の宣言は，ブロックCをのぞくブロックA全体に適用され，2番目の宣言はブロックCだけに適用される。したがって，ブロックBの中のGO TOステートメントが実行されると，ブロックAの中のL1に制御が移り，ブロックBが終了する。

DとBは，ブロックAの中で明示宣言されているので，Aの中でならばどこからでも参照することができる。しかし，これらはINTERNALなので，ブロックEの中で参照することはできない（Eへの引数として渡されない限り）。

```
                                   P    Q    Q'   R    R'   S    I
A: PROCEDURE;
   DECLARE P, Q;
   B: PROCEDURE;
      DECLARE Q;
      R = Q;
      C: BEGIN;
         DECLARE R;
         DO I = 1 TO 10;
         END;
         END C;
      END B;
   D: PROCEDURE;
      DECLARE S;
      END D;
   END A;
```

図 7.1　データ宣言の有効範囲

```
                                   L1   L1'  L2   A    B    C    D    E
A: PROCEDURE;
   DECLARE E ENTRY;
   L1: P = Q;
   B: PROCEDURE;
      L2: CALL C;
      C: PROCEDURE;
         L1: X = Y;
             CALL E;
             END C;
      GO TO L1;
      END B;
   D: PROCEDURE;
      END D;
   CALL B;
   END A;
E: PROCEDURE;
      END E;
```

図 7.2　入口宣言およびラベル宣言の有効範囲

Cは，Bの中で明示宣言されているので，Bの中から参照することはできるが，Bの外側から参照することはできない。

L2は，Bの中で宣言されているので，C（Bの中に含まれている）も含めてブロックBの中で参照することができるが，Bの外側から参照することはできない。

INTERNAL属性とEXTERNAL属性

INTERNAL属性をもつ名前の有効範囲は，その宣言の有効範囲と同じである。同じ名前に対して別の明示宣言が行なわれると，有効範囲の重複しない，別の新しい対象を指すことになる。

EXTERNAL属性をもつ名前は，同じプログラムの中で2回以上宣言することができる（別々の外部プロシージャーの中，または外部プロシージャーの中に含まれるいくつかのブロックの中のいずれでも可）。名前が宣言されるたびに，有効範囲が設定される。これらの宣言はいっしょに結びつけられ，1つのプログラムの中では，EXTERNAL 属性をもつ同じ識別名に対する宣言はすべて，同じ名前を指すようになる。名前の有効範囲は，プログラムの中でその名前に対して行なわれたすべての宣言の有効範囲を合計したものである。

注：PL/Iでは，データの外部名の長さは7桁を起えてはならず，また，_という文字（分割文字）を含んでいてはならない。

これらの宣言はすべて同じものを指しているので，与えられるいくつかの属性はすべて同じにならなければならない（ただし，INITIAL 属性が必要なときは，1回だけ指定する必要があるという点は例外）。コンパイラーがすべての宣言をチェックすることは不可能な場合もあるので（とくに，別のプロシージャーで名前が宣言されている場合は），EXTERNAL属性をもつ同じ名前に対する宣言はそれぞれ，合致する属性をもつよう注意しなければならない。このコンパイラーでは，任意選択の出力として属性リストをとることができるので，これを使用すれば，名前の使い方をチェックすることができる。次の例は，上で述べた点を，プログラムの中で例として示したものである。

```
A: PROCEDURE;
    DECLARE S CHARACTER (20);
DCL SET ENTRY(FLOAT DECIMAL), OUT
ENTRY(LABEL);
    CALL SET (3);
E: GET LIST (S,M,N);
    B: BEGIN;
        DECLARE X(M,N), Y(M);
        GET LIST (X,Y);
        CALL C(X,Y);
        C: PROCEDURE (P,Q);
         DECLARE P(*,*), Q(*),
             S BINARY FIXED EXTERNAL;
            S = 0;
            DO I = 1 TO M;
   IF SUM (P(I,*)) = Q(I)
            THEN GO TO B;
         S = S + 1;
         IF S = 3 THEN CALL OUT (E);
         CALL D(I);
        B: END;
           END C;
        D: PROCEDURE (N);
         PUT LIST ('ERROR IN ROW ',
              N, 'TABLE NAME ', S);
           END D;
        END B;
    GO TO E;
    END A;
OUT: PROCEDURE (R);
    DECLARE R LABEL,
        (M,L) STATIC INTERNAL INIT(0),
        S BINARY FIXED EXTERNAL,
        Z FIXED DECIMAL(1);
    M = M + 1; S = 0;
    IF M < L THEN STOP; ELSE GO TO R;
SET: ENTRY (Z);
      L = Z;
      RETURN;
      END OUT;
```

Aは，外部プロシージャー名で，その有効範囲は，ブロックA全体と，Aが外部名として宣言されている（DECLARE ステートメントによるか，またはプロシージャー名として用いることによって）他のブロックとを合せたものである。

Sは，ブロックAとブロックCの中で明示的に宣言されている。この文字ストリングの宣言は，ブロックCをのぞいたブロックA全体に適用される。ブロックCの中でだけは，固定2進数の宣言が適用される。Dは，ブロックCの中から呼び出されてはいるが，Dの中にあるPUTステートメントで参照されているSはブロックCで宣言されているSではなく，文字ストリングSを参照する。

Nは，ブロックDの中でパラメーターとして用いられているが，このブロックの外側でも使用されている。Nはパラメーターとして記入されているので，NはDの中で明示的に宣言されたことになる。また，Dの外側でNを参照すると，ブロックAの中でNが暗示的に宣言されることになる。Nという名前に対するこの2つの宣言は，この例では偶然，この2つの対象はともに，省略時解釈によって，FIXED（15，0），BINARYとINTERNALという属性をもってはいるが，別の対象を指すものである。

XとYは，B全体に知られているので，Bの中にあるブロックCまたはDの中で参照することはできるが，Bの外側のAの部分では参照できない。

PとQはパラメーターなので，パラメーター・リストの中にこれらを記入すれば，充分に明示宣言の働きもする。しかし，PとQが配列であることを指定するためには，別のDECLARE ステートメントが必要である。引数XとYは配列として宣言されているし，ブロックCの中で知られてはいるが，それでもなお，PとQも配列であることを指定するために，DECLAREステートメントの中でPとQを宣言する必要がある。（星印の表記は，パラメーターの限界が，引数の限界と同じであることを表わしている。）

IとMは，外部プロシージャーAの中で，明示宣言されていないので，これらは暗示的に宣言され，A全体に知られるようになる（IはブロックCの中でしか使われていないが）。

この例の2番目の外部プロシージャーは，SETとOUTという2つの入口名をもっている。これらは，ENTRYおよびEX－TERNAL属性をつけて明示的に宣言されているとみなされる。これらはまた，プロシージャーAの中でもENTRY属性をつけて明示宣言しなければならない。したがって，2つの入口定数SETとOUTは，2つの外部プロシージャー全体にわたって知られるようになる。

ラベルBは，このプログラムの中に2回出てきている。1回は，開始ブロックのラベルとして，すなわちAの中にあるラベルとして，明示宣言されている。次にまた，ENDステートメントの接頭語として用いられて，ブロックCの中のラベルとして再宣言されている。したがって，ブロックCの中にあるGO TOステートメントの中でBを参照すると，ブロックCの中のENDステートメントのラベルを指すことになる。ブロックCの外側で，Bを参照したときは，開始ブロックのラベルを指すようになる。

CとDは，Bの中であればどの地点からでも呼び出すことができ，るが，Bの外側のAの部分や別の外部プロシージャーからは呼び出すことができない。同じように，Eは外部プロシージャーA全体に知られているので，Aの中のどの地点からでも，Eに制御を移すことができる。

しかし，ブロックCの中のラベルBは，Cの中からしか参照することができない。GO TOステートメントによって，ブロックの外側に制御を移すことはできるが，ネストされたブロックへのこのような制御の移動は，通常はできない。これに対する例外が外部プロシージャーOUTの中に示されているが，ここでは，ブロックAからのラベルEが，ラベル・パラメーターRに対する引数として渡されている。

Eは，Aの中で宣言されており，OUTの中では知られていないが，GO TO R というステートメントは，ラベルEに制御を渡す。

変数MとLは，ブロックOUTの中で STATIC と宣言されている。これらの値は，OUTへの呼出しと呼出しの間で保存される。

プロシージャーOUTの中にあるSを，プロシージャーCの中にあるSと同じであると識別するために，この両方は，属性EXTERNALをつけて宣言されている。

外部構造のメンバー名の有効範囲

大構造名が2つ以上のブロックの中で EXTERNAL 属性をつけて宣言されているときは，対応するメンバーの名前は同じでなくてもよいが，対応する構造のメンバーの属性はそれぞれ，同じでなければならない。構造のメンバーの名前は，つねにINTERNAL属性をもっており，有効範囲の属性をつけて宣言することはできない。しかし，外部構造のメンバーを参照するとき，その参照を含んでいるブロックに知られているメンバー名を使って参照すれば，対応するメンバーの名前が同じであるかどうかには無関係に，その外部名が知られているすべてのブロックの中でそのメンバーを参照したのと，実質的に同じことになる。たとえば，

```
PROCA: PROCEDURE;
       DECLARE 1 A EXTERNAL,
                 2 B,
                 2 C;
          :
END PROCA;
PROCB: PROCEDURE;
       DECLARE 1 A EXTERNAL,
                 2 B,
                 2 D;
          :
END PROCB;
```

この例では，PROCAの中でA．Bが変更されれば，PROCBでも変更され，その逆もまた同じである。また，PROCA でA．Cが変更されれば，PROCB でもA．Dが変更され，その逆もまた同じである。

多重宣言と不明確な参照

同じブロックの内部にある同じ識別名について2つ以上の宣言が行なわれているときは，少なくともその1つが，その名前を唯一無二のものにするような名前の修飾を使うことができる形で構造の中で宣言されていない限り，それは多重宣言である。

同じ識別名に関して，異なる属性をつけて EXTERNAL 名として，プログラムの中のどこかで2度以上宣言すると，多重宣言になる（ただし，INITIAL属性はすべての宣言の中で指定する必要はないので例外）。

多重宣言は，エラーである。

修飾される名前は，その名前が唯一無二になる範囲で修飾をすればよい。ある名前を参照すると，つねにその参照を含んでいるもののうち最も内側にあるブロックで宣言されている識別名に適用されるものとみなされる。不明確な参照とは，その名前を唯一無二のものにするための修飾が不充分な名前である。

次にあげるいくつかの例は，多重宣言と不明確な参照の両方の例である。

```
DECLARE 1 A, 2 C, 2 D, 3 E;
   BEGIN;
   DECLARE 1 A, 2 B, 3 C, 3 E;
   A.C = D.E;
```

この例では，A．Cは内側のブロックのCを参照しており，D．Eは外側のブロックのEを参照している。

```
DECLARE 1 A,2 B,2 B,2 C,3 D,2 D;
```

この例では，Bは多重宣言されたことになる。A．Dは，2番目のDの場合にだけ修飾として完全なので，2番目のDを参照することになる。最初のDは，A．C．Dとして参照すればよい。

```
DECLARE 1 A,2 B,3 C,2 D,3 C;
```

この例では，Cはどちらも，この参照では完全に修飾されていないので，A．Cは不明確である。

```
DECLARE 1 A,2 A,3 A;
```

この例では，Aは最初のAを参照し，A．Aは2番目のA，A．A．Aは3番目のAを参照する。

```
DECLARE X;
DECLARE 1 Y,2 X,3 Z,3 A,
            2 Y,3 Z,3 A;
```

この例では，Xは最初の DECLARE ステートメントを参照している。

Y．Zへの参照は不明確である。X．Y．Zは2番目のZを参照しており，Y．X．Zは最初のZを参照している。

省略時解釈の属性の適用

PL／Iの原始プログラムでは，どのような識別名も，一揃いの属性を完全にもっていなければならない。しかし，PL／Iには，プログラマーがプログラムの中で使用するデータのタイプを容易に記述できるような機能がある。実際に，DECLAREステートメントの中で指定されている属性が，その識別名に必要な属性をすべて網羅している必要があることは殆んどない。さらに，文脈宣言の場合も，識別名に関して宣言されるのは，一部分だけである。このように，一部分の属性だけが宣言された識別名の各々について，コンパイラーは，省略時解釈の規則を適用して，暗示的に，一組の属性を完全にする。

PL／I言語およびシステム／360およびシステム／370の実施面で決められている省略時解釈の規則を，標準の省略時解釈の規則という。この標準の省略時解釈の規則を修正したり，完全な一揃いの新しい省略時解釈の規則を作りたい場合は，プログラマーは，かわりの省略時解釈の規則を定義することができる。DEFAULTステートメントは，このような目的に用いられるものである。その用法については，この章の後のセクションで説明する。

属性の適用の過程

コンパイラーによる属性の処理は，次のような順序で行なわれる。

1. 属性の因数化をはずす。
2. LIKE 属性を適用する。
3. 構造のメンバーに対して，ALIGNEDまたはUN-ALIGNED属性を適用する。
4. 明示宣言を設定し，適用する。
5. 文脈宣言を設定し，適用する。
6. 暗示宣言を設定する。
7. 明示的，文脈的，および暗示的に宣言された識別名に対して，DEFAULTステートメント（使用されていれば）で指定された属性を適用する。または，標準の省略時解釈の属性を適用する。
8. 属性の中で使用されている識別名，または別のプロシージャー・ブロックで宣言されている識別名も含めて，同一の識別名を解決する。

以上のことから，明示宣言または文脈宣言によって識別名に与えられた属性を，同じクラスの属性に関する省略時解釈の属性によって置き換えることはできないことが，わかるはずである。さらにまた，省略時解釈が適用される属性は，すでに適用された属性によって大きく左右される。このことは，DEFAULTステートメントの用法を理解する上で，重要なことである。

標準の省略時解釈の適用

標準の省略時解釈の属性は，特定のクラスの属性（たとえば，有効範囲，スケール，基数，またはモードなど）が，明示宣言によっても，文脈宣言によっても与えられていないときに，そのクラスの属性に対して適用される。

ファイル属性に対する標準の省略時解釈の要約は，第10章「入力と出力」のところにあげてある。問題データおよびプログラム制御データに対して想定される標準の省略時解釈の要約は，以下に示してある。標準の省略時解釈についての詳細は，I節「属性」のところをみていただきたい。

問題データ

問題データが，文字タイプであるか算術タイプであるかわからない場合は，算術タイプとみなされる。

算術データ：　そのデータに対して指定された情報によって，省略時解釈が異なる。

1. 算術データ項目について，明示宣言では一部分だけが指定されている場合は，省略時解釈によって次のような属性が想定される。

明示宣言	想定される省略時解釈の属性
BINARY	REAL, FLOAT
DECIMAL	REAL, FLOAT
FIXED	REAL, DECIMAL
FLOAT	REAL, DECIMAL
REAL	FLOAT, DECIMAL
FIXED BINARY	REAL
FIXED DECIMAL	REAL
FLOAT BINARY	REAL
FLOAT DECIMAL	REAL
REAL FIXED	DECIMAL
REAL FLOAT	DECIMAL
REAL BINARY	FLOAT
REAL DECIMAL	FLOAT

REALのかわりにCOMPLEXが指定されているときは，諸属性はREALの場合と同じであり，2つの部分の各々に適用されることに注意していただきたい。

2. スケールではなく基数が指定されている場合は，精度の属性の中のスケール因数の存在によって，スケールが決まる。スケール因数がある場合は，FIXEDとみなされ，ないときはFLOATとみなされる。たとえば，

```
DCL A BINARY(5),
    B BINARY(5,2);
```

Aに対して想定される属性は，REAL FLOATであり，Bの場合は，REAL FIXEDである。

3. モード，スケール，および基数が，DECLAREまたはDEFAULTステートメントによって，指定されていない場合は，識別名の頭文字によって属性が想定される。

頭文字	想定される省略時解釈の属性
$, #, @, A−H	REAL FLOAT DECIMAL
I − N	REAL FIXED BINARY
O − Z	REAL FLOAT DECIMAL

関数参照から返される値について，その基数，スケール，およびモードを決めるための省略時解釈の規則を設けることができる。返される値に対する省略時解釈の属性は，識別名の場合と同様，関数名に省略時解釈の規則を適用することによって，得られる。

算術データの精度： 算術データに対する標準の省略時解釈の精度は，以下の通りである。

属性	精度
FIXED BINALY	(15,0)
FIXED DECIMAL	(5,0)
FLOAT BINARY	(21)
FLOAT DECIMAL	(6)

算術データの他の属性： INTERNALの場合はALIGNEDおよびAUTOMATIC属性，EXTERNALの場合はSTATIC属性が想定される。

ストリング・データ： 文字またはビット・ストリングの長さが定義されていない場合は，長さは1とみなされる。INTERNALのときはUNALIGNEDとAUTOMATIC属性，EXTERNALのときはSTATIC属性が想定される。

構造および構造のメンバー： レベル1の構造は，INTERNALのときはAUTOMATIC，EXTERNALのときはSTATICとみなされる。小構造および構造のメンバーは，記憶域または有効範囲の属性をもたない。

配列およびデータ要素： ストリング・タイプのデータ要素に対しては，UNALIGNEDが想定される。これ以外のすべてのデータ・タイプについては，ALIGNEDが想定される。有効範囲および記憶域は，データのタイプによって異なる。

プログラム制御データ

入口： DECLAREステートメントの中，または外部ブロックのPROCEDURE／ENTRYステートメントのステートメント接頭語として宣言された入口定数は，EXTERNALとみなされる。入口変数は，INTERNALとみなされる。

ラベル，ポインター，オフセット，エリア，事象，入口の各変数またはファイル変数： これらの属性のいずれかをつけて宣言された識別名は，INTERNALのときはALIGNEDとAUTOMATIC，EXTERNALのときはSTATICとみなされる。エリア変数に対してサイズが指定されていないときは，省略時解釈によって1000バイトのサイズが適用される。

DEFAULTステートメント

DEFAULTステートメントは，識別名に割り当てられる属性に対してプログラマーが制御を加えることができるようにする働きをする。明示宣言または文脈宣言によって識別名に割り当てられた属性を置き換えるために，DEFAULTステートメントを使用することはできない。

DEFAULTステートメントは，標準の省略時解釈の規則を修正するため，またはプログラマーが定義した省略時解釈の規則の一まとまり全体を指定するために使用することができる。このステート

メントは，明示宣言または文脈宣言の後でも必要な属性が全部与えられていない識別名，入口宣言の記述子，およびPROCEDUREおよびENTRYステートメントのRETURNS オプションに対して，属性を指定することができる。プログラマーが定義した省略時解釈がプログラムの中に設定されたあとで，標準の省略時解釈の規則を復元することも可能である。

次に示すのは，DEFAULTステートメントの一般形式を簡潔にしたものである。

```
DEFAULT { RANGE(識別名｜文字：文字｜＊) }
        { DESCRIPTORS                   }
        〔属性の指定〕；
```

RANGEオプション： RANGEオプションは，関連する省略時解釈の規則が適用される識別名を指定するものである。コロンによって分離された２つの文字，または１つの識別名のどちらかとして範囲を指定することができる。たとえば，次のオプションは，

RANGE (A：J)

頭文字がAからJの範囲にあるすべての識別名に適用される。次のオプションは，

RANGE (ABC)

最初の３文字がABCであるようなすべての識別名（たとえば，ABC，ABCD，ABCDE など）に適用される。

RANGEオプションは，次のように指定することもできる。

RANGE (＊)

これは，AからZ，および$，@，および#という文字を頭文字とするものすべてを指定している。

DESCRIPTORSオプション： DESCRIPTORSオプションは，関連する省略時解釈の規則が，ナル以外のパラメーター記述子に適用されることを指定するものである。

属性の指定： 属性の指定とは，指定された範囲にある識別名に，選択された属性を適用するための属性のリストである。リストの中の属性はどのような順序になっていてもよいが，ブランクで区切らなければならない。

データ項目の宣言を完成するのに必要な属性だけが，属性のリストからとられる。必要なすべての属性がこのリストから得られないときは，標準の省略時解釈の属性が適用される。したがって，標準の省略時解釈と同じ属性は指定する必要がない。たとえば，

DEFAULT RANGE(T) POINTER；

これは，Tという文字で始まる識別名がポインターであることを意味している。これらの識別名に適用される属性のリストの内容は，POINTER，AUTOMATIC，INTERNAL，およびALIGNEDである。

データ項目に適用されたときに矛盾する属性が，属性の指定の中にあるときにも矛盾しているとは限らない。たとえば，

DEFAULT RANGE(S) BINARY VARYING；

これは，Sという文字で始まり，BITまたはCHARACTER属性をつけて明示宣言されている識別名には，VARYING 属性が与えられることを意味している。そのほかのもの（算術データ以外のものとして明示的または文脈的に宣言されていないもの）には，BINARY属性が与えられる。

VALUEオプションは，１０進整数の定数または式によって表わされる属性を指定するために，用いられる。このようなものには，長さ，サイズ，および精度の属性がある。たとえば，

DEFAULT RANGE(＊) VALUE(AREA(2000))；

このステートメントは，すべてのエリア変数に関する省略時解釈のサイズを２０００とするものである。次元の属性は，リストの最初に記入すれば，属性指定の中で直接指定することができる。

例　1：

次のような頭文字の範囲が，それぞれ右側に与えられている属性に対応するものとしよう。

頭文字	必要な属性
A − D	REAL FLOAT DECIMAL
E − H	REAL FLOAT BINARY
I − N	REAL FIXED BINARY
O − Z	REAL FIXED DECIMAL

精度として想定されるのは，IBMシステム／３６０の実施面で取り決められている省略時解釈の精度である。これらの追加の省略時解釈の規則を設定するためのDEFAULT ステートメントは，次のようになる。

```
DEFAULT RANGE(E:H) BINARY,
RANGE(O:Z)FIXED;
```

このステートメントの中では，２つの範囲の頭文字が，追加の省略時解釈の規則として指定されている。E−HおよびO−Zの範囲外の頭文字をもつ識別名に関する標準の省略時解釈の規則は，変らない。

例　2：

暗示宣言されたすべてのデータが指定されたものと同じ属性をもつことを，DEFAULT ステートメントで指定することができる。

```
DEFAULT RANGE(*) PICTURE '99999';
```

このステートメントは，暗示宣言されたすべての識別名は，PICTURE'99999' という属性をもつ数字タイプであると想定するためのものである。

標準の省略時解釈以外の値が必要な場合は，特定の省略時解釈の属性に対する精度，ストリングの長さ，またはエリアのサイズを規定するための属性を，VALUE オプションの引数につねに入れておかなければならない。たとえば，

a. DEFAULT RANGE (S:T) CHARACTER VALUE (CHARACTER (10));
b. DEFAULT RANGE (*) VALUE (FIXED BINARY(31), FLOAT DECIMAL(16), FLOAT BINARY(53), FIXED DECIMAL(15));

最初の例は，暗示宣言された，頭文字がSとTの識別名にはすべて，CHARACTERという省略時解釈の属性と，10桁の長さをもつストリングという省略時解釈を与えるものである。2番目の例は，精度が定義されていない算術タイプのすべての識別名は，VALUE というキーワードに対する引数の中で定義されている精度をもつことを指定するものである。（この例で指定されている精度は，許される最大の精度のものである。）

標準の省略時解釈の復元

次のステートメントは，

```
DEFAULT (RANGE(*) DESCRIPTORS);
```

外側のブロックの中で設定されているプログラマー定義の省略時解釈の規則と置き換わるものである。これは，外側のブロックの中にある省略時解釈の指定をいずれも置き換えてしまうので，標準の省略時解釈の規則を，現在のブロックの中の明示宣言に戻すために使用することができる。

DEFAULTステートメントの有効範囲

前にプログラマーが定義した省略時解釈の規則を復元したり，修正したりしているような，内側のブロックの中で DEFAULT ステートメントが指定されているのでない限り，DEFAULTステートメントの有効範囲は，それが指定されているブロック，およびそのブロックに含まれているブロックになる。たとえば，

```
A:PROC;
DEFAULT RANGE(A:I) FIXED BINARY;
      .
      .
      .
B:PROC;
DEFAULT RANGE(I) DECIMAL;
      .
      .
      .
END A;
```

プロシージャーBの中では，Iで始まる識別名の場合は，DECIMAL が BINARY に置き換わり，FIXEDは受けつがれない。境界合せ，有効範囲，記憶域クラス，および精度については，標準の省略時解釈も適用される。

内部ブロックに対して DEFAULT ステートメントが与えられている場合は，その有効範囲には，明示宣言された識別名だけを含むことができる。これは，文脈的および暗示的に宣言された識別名の有効範囲は，その名前が出ている外部プロシージャーの PROCEDUREステートメントの直後にある DECLARE ステートメントの中で宣言された場合と同じようにして決められるためである。

省略時解釈指定の因数化

省略時解釈の指定は，因数化することができる。たとえば，次のステートメントは，

```
DEFAULT (RANGE(A:C) FIXED, RANGE
(D:F) FLOAT) DECIMAL;
```

暗示宣言された算術的な識別名のうち，頭文字がAからCまでのものには，FIXED DECIMAL という属性が与えられ，頭文字がDからFまでのものには，FLOAT DECIMAL という属性が与えられることを指定している。

パラメーターの記述子に対するプログラマー定義の省略時解釈

パラメーターの記述子に対して属性を指定するために，DEFAULT ステートメントを使用することができる。DESCRIPTORS というキーワードは，パラメーターの記述子に対する属性の指定として，その後に属性のリストが続くことを指定するものである。たとえば，

```
DEFAULT DESCRIPTORS BINARY;
DCL X ENTRY (FIXED, FLOAT);
```

上の例では，リスト内にある各々のパラメーター記述子に対して BINARY という属性が追加され，次のリストと同じものが作り出される。

```
(FIXED BINARY, FLOAT BINARY)
```

DESCRIPTORSの省略時解釈の属性は，ナル記述子をもつパラメーター（すなわち，パラメーター記述子の中で属性が何も指定されておらず，したがって，その属性は対応する引数の属性と同じでなければならないようなパラメーター）には，適用されない。

RETURNSオプションに対するプログラマー定義の省略時解釈

関数プロシージャーから返される値のうち，暗示宣言されているものの属性は，そのプロシージャーを呼び出すのに用いられた入口名にしたがって決められる。DEFAULTステートメントを用いて，このような属性を指定することもできる。たとえば，次のステートメントは，

```
DEFAULT RANGE (X) FIXED BINARY;
X : PROC(Y);
```

以下のように解釈される。

```
X : PROC(Y) RETURNS (FIXED BINARY);
```

DEFAULTステートメントの用法に関する制約

DEFAULTステートメントで，ENTRY，ENVIRONMENT，RETURNS，LIKE，VARIABLEの諸属性，またはFILE以外のファイル属性を，指定してはならない。構造を指定するために，このステートメントを使用することはできないが，構造の要素の属性には，RANGEに応じて，省略時解釈を適用することができる。

明示的に宣言されていない識別名に対して，DEFAULTステートメントで次元属性を指定することができるが，添字をつけた識別名は，BUILTIN属性をつけて文脈的に宣言されることになる。したがって，省略時解釈によって次元属性を適用するのは，明示宣言された識別名に対してだけに限るべきである。たとえば，

```
DEFAULT RANGE (ARRAY) (10,10)
FIXED BINARY;
DCL ARRAY1, ARRAY2;
```

ARRAY 1とARRAY 2は両方とも，100個の要素からなる。明示宣言された2次元の配列であり，その要素はそれぞれ，FIXEDおよびBINARYという属性をもっている。

第8章： 記憶域の制御

この章では，どのようにすれば，PL/Iのプログラマーは，記憶域の割振りを制御することができるかということを解説する。割振りとは，変数に対して記憶域を取得する課程をいう。変数の世代とは，その変数に対する特定の割振りを指している。記憶域クラスには，STATIC(静的)，AUTOMATIC(自動)，CONTROLLED(被制御)，およびBASED(基底付)の4つがあり，プログラマーは，特定のプログラムでの必要に応じて，制御を行なうことができる。あらゆる変数に，記憶域が必要である。このことは，ストリングや算術などの問題データにも，ラベル変数や入口変数，ファイル変数などのプログラム制御データにも適用される。変数の宣言には，必ず記憶域クラスの属性を含めなければならない(省略時解釈によってだけでも)。変数の名前は，実際には変数のアドレスであり，変数に対して指定された属性は，必要な記憶域の量や，その解釈の仕方を記述するものである。たとえば，

```
DECLARE X FIXED BINARY(31,0) AUTOMATIC;
```

Xという名称は，フルワードを占める値，すなわち，固定小数点2進整数として解釈される4バイトの値をアドレスしている。静的変数や自動変数の場合は，この概念はさして重要ではないが，被制御変数，とくに基底付変数について考えるときは，この概念は非常に関係をもってくる。

PL/Iでは，変数に対する記憶域の割振りはコンパイラーが行なうので，プログラム内のどこをとっても，プログムマー自身が主記憶域内における変数の絶対アドレスをアクセスすることはない点を理解しておく必要がある。したがって，プログラマーが記憶域の割振りに関して行なう制御は，記憶域の必要量とその記憶域を割り振る時点に関することだけであり，主記憶域内のどこに割り振るかということは指定しない。ただし，エリア変数の中で基底付変数を割り振るときは，すでに割り振られている記憶域から相対的にみてどこに割り振るかを指定することができる。

どの程度に記憶域の制御をすることができるかということは，使用される記憶域のクラスによって異なる。

静的記憶域

STATIC 属性をつけて宣言されている変数は，プログラムが実行される前に割り振られ，プログラムが終了するまでそのまま割り振られておかれる。実行中に，プログラムで静的変数の割振りを制御する手段はない。多くの場合プログラムの中では，実行中にいつでも使用されるデータが必要である。たとえば，プログラムの中で指定されるすべての算術定数は，STATIC が宣言されている変数と同じようにして貯えられる。両者の相違は，静的変数の値はプログラムの実行中に変えることができるのに対して，定数は変えることができないことである。静的変数は，プログラム内のどこでも宣言することができるが，これはみな，実行の前に割り振られる。しかし，静的変数への参照を有効なものにするためには，通常の有効範囲の規則にしたがうよう，充分注意しなければならない。たとえば，

```
A:PROC OPTIONS(MAIN);
    ⋮
  B:PROC;
    DECLARE X STATIC INTERNAL;
    ⋮
    END B;
  END A;
```

変数Xは，プログラムの間中割り振られてはいるが，これを参照できるのは，プロシージャーBの中，またはBの中に含まれるブロックの中だけである。

INITIAL 属性を用いて静的変数を初期値化する場合は，ポインター変数の場合をのぞいて(これについては後でのべる)，初期値を定数として指定しなければならない。また，範囲の指定(たとえば，配列の限界)も定数でなければならない。したがって，静的記憶域を使用する場合は，プログラムを書くときに指定した割振りは，それがどのようなものであろうとも，プログラム実行の間中そのままになっていることを，つねに心に留めておかなければならない。プログラム内のどこからも必要とされるようなデータはすべて，静的記憶域を使用しなければならない。STATIC ポインターまたはオフセット変数は，NULL 組込み関数によってのみ，初期値化することができる。

この他の形式の記憶域の割振りはすべて，動的である。つまり，プログラムの実行中に記憶域が得られる。このようなことから，プログラマーは，静的変数の場合以上に制御を働かせることができる。

自動記憶域

AUTOMATIC属性をつけて宣言されている変数は，その変数が宣言されたブロックに入るときに割り振られる。このような変数は，プログラムの実行中に何度でも割振りし直すことができる。
プログラマーは，プログラムのブロック構造の設計の仕方如何によって，その割振りを制御する。たとえば，次の例では，

```
A:PROC;
   :
   CALL  B;
   B:PROC;
      DECLARE  X, Y  AUTO;
      :
      END  B;
   :
   CALL  B;
```

プロシージャーBが呼び出されるたびに，変数XとYに記憶域が割り振られ，Bが終了すると，記憶域が解放される。したがって，それらがもっていた値は失なわれる。解放された記憶域は，他の変数への再割振りに使用することができる。したがって，ブロック（プロシージャーまたは開始）が活用化されているときはつねに，そのブロックの中で自動変数として宣言されているすべての変数に記憶域が割り振られており，ブロックが活用化されていないときはつねに，そのブロック内にある自動変数に記憶域は割り振られていない。RECURSIVEオプションをもつプロシージャーの場合をのぞいて，ある特定の自動変数に対する割振りは，1つしか存在しない。

自動変数に対する配列の限界，ストリングの長さ，およびエリアのサイズを，式として指定することができる。このことは，必要なときに記憶域を割り振ることができるということばかりでなく，ほんとうに必要な量の記憶域を割り振ることができるということも意味している。たとえば，

```
A:PROC;
   DECLARE  N  FIXED  BIN;
   :
   B:PROC;
      DECLARE  STR  CHAR(N);
```

文字ストリングSTRは，プロシージャーBが呼び出されたときに存在していた変数Nの値によって定義されている長さをもつことになる。しかし，このような式を評価するために余分な操作が必要になるので，実行の速度をある程度犠牲にして記憶域が保存される。

自動変数における回帰の影響

すでに活用化されているときに呼び出すことのできるプロシージャーを回帰的という。このようなプロシージャーの中で特定の活用化に対して割り振られた変数の値は，他の活用化から変えられないように保護しなければならない。このような保護は，積重ねと呼ばれる処理法を用いて，他の呼出しの値を参照することができないようにすることによって行なわれる。積重ねは，後入先出方式で操作される。ある時点で参照可能な自動変数は，最新の世代の自動変数だけである。静的変数は，回帰の影響を受けないことに注意していただきたい。したがって，静的変数は，回帰の状況のもとで，回帰的な呼出し相互の間の連絡に用いると有効である。このことはまた，回帰的なプロシージャーを含んでいるプロシージャーの中で宣言されている自動変数についても，被制御変数と基底付変数についてもいえる。たとえば，

```
A:PROC;
   DCL  X;
   :
      B:PROC  RECURSIVE;
      DCL  Z,
           Y  STATIC;
      CALL  B;
      :
      END  B;
END  A;
```

変数Xの場合は，1つの世代だけが，プロシージャーBの呼出しの間中存在している。変数Zは，プロシージャーBを呼び出すたびに別の世代をもつことになる。変数Yは，プロシージャーBの中でだけ参照可能であり，呼出しのたびに再割振りされることはない。（変数の積重ねの概念は，被制御変数の説明のときにも重要なことである。）

被制御記憶域

CONTROLLEDと宣言されている変数は，ALLOCATEステートメントの中でそれらが指定されたときにだけ割振られる。
被制御変数の各々に対して，プログラマーが個々に制御をする。被制御変数の有効範囲は，内部的と宣言されている場合は，それが宣言されているブロックとその中に含まれているブロックである。被制御変数の宣言では，変数を割り振るときに記憶域がどれだけ必要か，またそれをどのように解釈するかということに関する属性だけを記述する。たとえば，

```
A:PROC;
  DCL X CONTROLLED;
  :
  CALL B;
  :
      B:PROC;
  ALLOCATE X;
      :
      END B;
 END A;
```

変数Xは，プロシージャーB，およびプロシージャーAのうちCALLステートメントの後にある部分の中では，正しく参照することができる。CALLステートメントの前で，この変数の値を参照してもエラーになる。一旦被制御変数が割り振られたあとは，その変数を指名するEREEステートメントに出会うか，またはプログラムが終了するまで，割り振られたままの状態になっている。被制御変数の有効範囲は，プログラム全体でないこともある点に注意していただきたい。これは，前に「静的記憶域」の項で説明したSTATIC INTERNAL変数の場合と似た状況になる。つまり，存在してはいるが，参照できないわけである。

FREEステートメントは，被制御変数に対して割り振られた記憶域を解放するものである。このあと，記憶域は，他の割振りをするために再使用することができる。

一般に，範囲が調節できるような，大きなデータ集合体がプログラムの中で必要な場合に，被制御変数は有益である。たとえば，

```
DCL A(N,M) CTL;
  :
GET LIST(M,N);
ALLOCATE A;
GET LIST(A);
  :
FREE A;
  :
```

このプログラムの手順は，入力データの状況に応じて本当に必要な量の記憶域を割り振り，不要になった時点でデータを廃棄する（その記憶域を解放する）。この方法の場合は，エピローグもプロローグも不要であるから，開始ブロックを設定するもう１つの方法よりも，効率がよい。

被制御変数に対するALLOCATEステートメント

被制御変数は，ALLOCATEステートメントによってのみ，割り振ることができる。被制御変数に対するALLOCATEステートメントの一般形式は，以下の通りである。

```
ALLOCATE 〔レベル〕識別名
         〔次元属性〕〔属性〕
         〔,〔レベル〕識別名
         〔次元属性〕
         〔属性〕〕…
         〔INITIAL〕;
```

「識別名」は，CONTROLLED属性をもつ変数である。これは，要素，配列，または構造のいずれであってもよいが，添字をつけたり，修飾したりすることはできない。属性として指定できるのは，次元の範囲，ストリングの長さ，エリアのサイズを指定するものだけである。（エリアについては，この章の後の方で説明するが，この文脈では，これは単に，記憶域の調節が可能な変数として扱う。）このことによって，プログラマーは，変数の特定の世代ごとに記憶域の量を変えることができるようになる。このような属性には，次のものがある。

次　元
CHARACTER（長さ）
BIT（長さ）
AREA（サイズ）

次元属性は，他のいずれとでもいっしょに指定することができる。たとえば，

```
DCL X(20) CHAR(5) CONTROLLED;
  :
ALLOCATE X(25) CHAR(6);
  :
```

ALLOCATEステートメントの中で指定されている属性の値はつねに，同じ変数に対してDECLAREステートメントの中で与えられているものを置き換える。しかし，属性自体は一致していなければならない。したがって，次元属性は，同じ数の次元を指定しなければならない。DECLAREステートメントの場合と同様，要素式を使って限界，長さ，およびサイズを指定することができる。

ALLOCATEステートメントの中で，INITIAL属性も指定することができる。ALLOCATEステートメントの中で与えられた初期値が，DECLAREステートメントの中の初期値（指定されていれば）と置き換わる。

被制御変数に対するFREEステートメント

被制御変数を指名しているFREE ステートメントが実行されると，その変数の記憶域が解放され，その値が失なわれる。FREE ステートメントの形式は，次の通りである。

FREE　識別名〔，識別名〕…；

この「識別名」には，ALLOCATEステートメントの場合と同じ制約がある。

FREE ステートメントが指定している変数が，割振りの行なわれなかったものである場合は，何の処置もとられない。

暗示的な解放

プログラムが終了するまで，被制御変数を割り振られたままにしておく場合は，FREE ステートメントによって明示的に解放する必要はない。プログラムが終了すると，すべての被制御記憶域は，自動的に解放される。

被制御変数の複数個の世代

被制御変数の記憶域が，解放される前に再割振りされると，最初の世代が保存される（すなわち，積み重ねられる）。2番目の世代が現在の世代になり，最初の世代は，現在の世代が解放されるまで，アクセスできなくなる。これは，回帰的プロシージャーのところで自動変数について述べた処理法と似ている。しかし，被制御変数の場合は，変数の積重ねおよび積重ねからの解放は，ブロックの境界のところではなく，ALLOCATEおよびFREE ステートが与えられたところで起こるようになっており，プロシージャーの呼出しとは無関係である。

被制御変数の連続する世代の値は，積重ねられるが，この値を最新の世代からとり出して，新しい世代を作るのに役立てることができる。ALLOCATEまたは DECLARE ステートメントの中で，境界，長さ，またはサイズが，その変数への参照を含んでいる式によって指定されている場合は，その前の最も新しい世代から値がとられる。たとえば，

```
DCL X(20)FIXED BIN CTL;
   ⋮
ALLOCATE X;
   ⋮
ALLOCATE X(X(1));
```

Xの最初の割振りの中では，上限は，DECLARE ステートメントによって指定される（つまり，20になる）。2番目の割振りでは，上限は，Xの最初の世代の最初の要素の値によって指定される。

星印の表記

ALLOCATE ステートメントの中で，次元，長さ，またはサイズが星印によって表示されている場合は，その値は，その前の最も新しい世代から受けつがれる。配列の場合は，配列のどの次元に対しても（1つだけでなく），星印を使わなければならない。

```
DCL X(10,20) CHAR(5) CTL;
   ⋮
ALLOCATE X;
   ⋮
ALLOCATE X(10,10);
   ⋮
ALLOCATE X(*,*);
```

この例では，Xの最初の世代の限界は，（10，20）であり，2番目と3番目の世代の限界は，（10，10）である。Xの各世代の要素はすべて，長さが5の文字ストリングである。

DECLAREステートメントの中でも星印の表示を用いることができるが，これは別の意味をもっている。たとえば，

```
DCL Y CHAR(*)CTL,
         N FIXED BIN;
         N=20;
         ALLOCATE Y CHAR(N);
```

これは単に，文字ストリングYの長さを，ALLOCATE ステートメントの中で指定することを意味している。この方法を使うと，プログラマーは，記憶域が実際に割り当てられるまでストリングの長さの指定を延期することができる。

被制御構造

構造を被制御にすると，その中に含まれる配列，ストリング，またはエリアはいずれも，調節可能になる。このような理由から，ALLOCATE ステートメントの中で相対的に構造の記述ができるようになる。たとえば，

```
DCL 1 A CTL,
      2 B(−10:10),
      2 C CHAR(*) VARYING;
      :
ALLOCATE 1 A,
           2 B,
           2 C CHAR(5);
      :
FREE A;
```

構造が割り振られると，A．Bの範囲は－10から＋10になり，A．Cは，最大の長さが5で，値がナルの，長さの変わる文字ストリングになる。この構造が解放されるときは，大構造名だけが与えられる。被制御構造のすべてをいっしょに解放するか，割り振るかしなければならない。構造の一部分に対して記憶域をとっておくことは，エラーである。

組込み関数の割振り

被制御変数の現在の状況－"割り振られている"か"解放されているか"－が，プログラムが通る経路次第で決まる場合は，変数が割り振られているかどうか判断できるようにすると便利である。ALLOCATION組込み関数は，その被制御変数に対する存在の中で，世代の番号を示す2進数値を戻す。変数が割り振られていないときは，ゼロの値が戻される。関数参照は，次のような形式をしているが，

```
ALLOCATION(a)
```

aは，被制御変数でなければならない。

ALLOCATION組込み関数のほかにも，役に立つ組込み関数がいくつかある。このような関数には，配列に対して指定された次元の範囲を判定する配列取扱い関数DIM，所与の配列に指定された次元の上限と下限をそれぞれ判定するLBOUNDとHBOUND，がある。同様に，ストリングに対しても，ストリングの現在の長さを返す組込み関数LENGTHがある。

基底付記憶域

基底付変数と，その他のすべての記憶域クラスとの基本的な相違は，基底付変数の名前は，主記憶域内における世代の位置を識別しないということである。したがって，基底付変数の宣言では，その世代を記述するだけ，すなわち記憶域の必要量と記憶域の解釈の仕方を記述するだけである。その世代の位置は，ロケーター変数と呼ばれる別の変数によって識別される。ロケーター変数は，ポインター変数またはオフセット変数のいずれかである。オフセット変数については，エリア変数と結びつけて，本章の後の方で説明する。

被制御変数の宣言も，記憶域を記述するだけであるが，その変数に対して一旦ALLOCATE ステートメントが実行されたあとは，その名前は，変数の位置も識別するようになる。このような理由から，プログラム内の特定の一地点で，被制御変数の2つ以上の世代を参照することは不可能である。事実，基底付変数に対してALLOCATE ステートメントを使用することもできるが，どの世代の位置も独立したロケーター変数によって識別されるので，適当なロケーター値を使用すれば，プログラム内のどの地点からも，基底付変数のどの世代でも参照することができる。

基底付変数

基底付変数の宣言では，BASEDというキーワードと，任意選択で，その基底付変数と結びつけられるロケーター変数の名前を指定する。たとえば，

```
DCL X FIXED BIN BASED(P);
```

この宣言の場合，変数Pの値によって，変数Xの位置が識別される。

このようなポインター変数の結びつけ方は，特別な関連づけではない。Pを用いて，他の基底付変数の位置を識別することができるし，また，他のロケーターを用いてXの他の世代を識別することもできる。

ロケーターの修飾

基底付変数の値の参照は2つの部分からなっているので，これは修飾された参照になり，構造のメンバーへの参照と区別するために，ロケーター修飾参照と呼ばれる。合成記号－>（負符号のすぐ後に「より大」の記号をつづけて書く）は，「により修飾された」または「～を指す」ということを表わしている。たとえば，

```
P -> X
```

Xは基底付変数，Pはロケーター式でなければならない。この参照は，ロケーターPの値によって識別されるXの世代を意味している。

基底付変数が，宣言の中でロケーター変数と結びつけられている場合は，プログラマーは，参照の中では，基底付変数の名前だけを指定すればよい。たとえば，

```
DCL X FIXED BIN BASED(P);
      :
ALLOCATE X;
      :
X = X + 1;
```

(このALLOCATEステートメントは，Xの参照が，割り振られた記憶域に適用されるように，ポインター変数Pの値をセットする。)割当てステートメントの中でXを参照すると，暗示的にPによってロケーター修飾される。次のように，明示的に参照をロケーター修飾することもできる。

```
P->X = P->X + 1;
```

この割当てステートメントは，前の例と同じ効果をもっている。基底付変数は，ポインター変数を指名せずに宣言することができる。このような場合は，基底付変数を参照するときはつねに，明示的にロケーター修飾しなければならない。

(PL/Iでは，ここで述べたものよりももっと一般的なロケーター修飾の形式が許されていることに注意していただきたい。このことについては，この章の終りの方の「複数個のロケーター修飾」のところをみていただきたい。しかし，この章の残りの部分を理解する上で，この一般形式はあまり重要ではない。)

ポインター変数

ポインター変数は，それが基底付変数の宣言の中に出てきたとき，ロケーター修飾として出てきたとき，あるいはALLOCATE，LOCATE，またはREADステートメントのSETオプションの中に出てきたときに，文脈的に宣言される。これはまた，次の例の場合のように，明示的に宣言することもできる。

```
DCL Q POINTER;
```

Qは変数なので，記憶域クラスをもっていなければならない。この場合には，省略時解釈によりAUTOMATICが適用される。ポインター変数はプログラム制御変数であるから，算術値と同じように扱うことはできない点に注意していただきたい。ポインター変数を集めて，配列や構造にすることができる。

ポインター式

ポインター式とは，ポインター変数(修飾または添字をつけることができる)，またはポインター値を返す関数参照のいずれかである。

ポインター式は，次のようにして使用することができる。

1. ロケーター修飾として(基底付変数の宣言と結びつけて)。
2. 比較演算の中で，たとえば，IFステートメントの中で(ポインター値は，等，不等の比較しかできない)。
3. プロシージャー参照の中の引数として。

ポインター変数のセット

ポインター修飾変数を参照する前に，ポインターに値を与えておかなければならない。ポインター値は，次のいずれかから得ることができる。

1. NULL 組込み関数
2. ADDR 組込み関数
3. READ またはLOCATE ステートメント
4. ALLOCATEステートメント

すべてのポインター値は，最初は，これらの4つの方法のいずれかからとられる。そのあとで，このような値は，ポインター変数にポインター値を複写する割当ての方法や，オフセット値をポインター値に変換する(あるいはその逆の変換)ロケーター変換の方法，プロシージャー参照の引数としてポインター値を渡す方法，関数プロシージャーからポインター値を戻す方法，によって処理することができる。

ADDR組込み関数

ADDR組込み関数は，変数の最初のバイトを識別するポインター値を返す働きをする。この変数は，どのようなデータ・タイプまた

は構成のものでよいし，どのような記憶域クラスのものでもよい。たとえば，

```
P = ADDR(X);
```

ここで，Pは，ポインター変数であり，Xは連結される変数である。この組込み関数への引数は，添字付の修飾された参照であってもよい。たとえば，

```
DCL A(3,2) CHARACTER(5) BASED(P),
    B CHAR(5) BASED(Q),
    C(3,2) CHARACTER(5);
    :
P = ADDR(C);
Q = ADDR(A(2,1));
```

この例では，配列AおよびCは，同じ記憶域を参照している。要素BとC(2，1)も同じ記憶域を参照している。

このような方法で基底付変数がオーバーレーされるときは，新しい記憶域は割り振られないことに注意していただきたい－－基底付変数は，基底付変数がその上にオーバーレーされる変数(この例ではA(3，2))と同じ記憶域を使用する。

このオーバーレーの技法は，DEFINED属性を用いて行なうこともできるが，DEFINED属性の場合のオーバーレーは永久的であるという点が，重要な違いである。基底付変数がオーバーレーされるときは，ポインター変数に新しい値を割り当てることによって，プログラムの中でいつでも，その結びつきを変えることができる。普通，PL/Iでは，データ・タイプの異なる変数でオーバーレー(たとえば，ビット・ストリングによって整数をオーバーレー)することは許されないが，このコンパイラーでは，特別に可能になっている。ただし，基底付変数の属性とオーバーレーされる変数の属性との間の矛盾を認識する手段をコンパイラーがもっているわけではないことを理解しておいていただきたい。

ADDR組込み関数では，変数の構成についての情報は与えられない。したがって，変数が集合体であるときに，全体として参照したければ，連続する記憶域に入れておかなければならない。たとえば，変数が配列の断面である場合は，それらの要素をインターリーブしてはならない。しかし，このコンパイラーでは，変数が長さの変わるストリングまたはエリアであるときは，制御情報は変数の必須の部分である。長さの変わるストリングの前には，2バイトの長さフィールドがつけられており，エリアの前には16バイトの制御情報がつけられている。したがって，これらのタイプの変数に対してADDR関数が実行されたときは，ポインター値は制御情報の先頭を指している。

ADDR関数の使い方に関するその他の規則は，G節「組込み関数と擬似変数」のところで述べる。

基底付変数と入出力

基底付変数は，ストリーム方式またはレコード方式転送のいずれかを用いて，転送することができる。

リスト指示形式のストリーム方式転送では，基底付変数がロケーター修飾(暗示的または明示的に)されていれば，他のタイプの変数と同じようにして処理される。たとえば，

```
GET LIST (P->X);
```

しかし，データ指示転送では，宣言の中でロケーター式と結びつけられた基底付変数だけを転送することができる。たとえば，

```
DCL Y BASED(Q), Z BASED;
:
PUT DATA(Y);
```

変数Zは，PUT DATAまたはGET DATAステートメント(つまり，データ指示の入出力)では，転送することができない。ストリーム方式の転送の技法と機能については，第11章で説明する。

レコード方式の転送には，2つの処理モードがある。移動モードは，緩衝域を使って，変数の割り振られた世代の中へ(またはその中から)直接的または間接的にデータを移動するものである。位置指定モードは，緩衝域へ(または緩衝域から)データを移動し，緩衝域の中に割り振られた記憶域を識別するだけの働きをする。基底付変数は，どちらのモードを用いても転送することができるが，本来は，位置指定モードを使用するよう設計されたものである。位置指定モードでは，基底付変数は緩衝域の内容を記述するために用いられるので，データが緩衝域の中にある間は，処理することができる。

位置指定モードは，BUFFEREDファイルに対してだけ適用されることに注意していただきたい。また，このファイルは，キーの順序になっているVSAMデータ・セットに関連づけられたINPUTおよびUPDATEファイルの場合以外はSEQUENTIALでなければならない。

第12章「レコード方式の転送」のところで，この2つのモードについて，詳しく説明する。

SETを伴なうREAD ステートメント

位置指定モードでは，READ ステートメントは次のような形式をしている。

```
READ FILE(ファイル式)
SET(要素ポインター変数);
```

このステートメントは，緩衝域にレコードを入れ，指定されたポインター変数をセットすることによって，その位置を識別する。このポインター変数によって修飾された基底付変数はいずれも，緩衝域の内容を記述する。たとえば，

```
DCL X CHAR(20) BASED(P),
      Y(20) CHAR(1) BASED(P);
              ⋮
READ FILE(IN) SET(P);
              ⋮
```

このプログラム・セグメントでは，1つのレコードが緩衝域に読み込まれ，ポインター変数Pがその位置を識別する。緩衝域の中のレコードは，基底付変数Xによって，固定長文字ストリングとして，また基底付変数Yによって，1桁の文字の配列として，同時に取り扱われる。Pは，文脈的に，ポインター変数と推論されること，XまたはYへの参照は，暗示的にPによって修飾されることに注意していただきたい。

LOCATEステートメント

LOCATE ステートメントは，SETを伴なうREADステートメントを補うものであって，緩衝域からの出力に対して用いられる。

```
LOCATE 基底付変数
FILE(ファイル式)
〔SET (要素ポインター変数)〕;
```

このステートメントは，指定された基底付変数に対して緩衝域の中に記憶域を割り振るものである。SETオプションは，基底付変数が，宣言の中でポインター変数と結びつけられていないときにだけ指定すればよい。

LOCATE ステートメントは，他のすべての転送ステートメントとは異なる働きをする。このステートメントは，記憶域アドレスを指すポインターをセットするものであるから，その記憶域への値の割付けが終了するまで，転送するものは何もない。LOCATE ステートメントは，前のレコード(つまり，前のLOCATE ステートメントによって得られた記憶域の内容)を転送し，そのレコードに対する記憶域を解放して，その記憶域を次のレコードに割り振る。同じファイルに対してWRITEまたはCLOSEステートメントが実行されたときにも，現在のレコードが転送される。次の例は，LOCATE ステートメントの用法を示すものである。

LOCATE ステートメントの操作方式の性格から，いくつかのファイルをグループにしている場合は，装置と関連づけられたファイルに対してこの方式を使うときは，注意する必要がある。このような場合は，グループ内のいずれかが閉じられたあとは，転送を行なうことができない。(第12章「装置と関連づけられたファイル」を参照のこと)。

```
   DCL 1 STR BASED(P),
         2 NAME CHAR(20),
         2 RATE FIXED(5,2);
           ⋮
OUTPUT:LOCATE STR FILE(OUT);
           ⋮
  /*ASSIGN VALUES TO STR*/
           ⋮
   GO TO OUTPUT;
```

位置指定モードを用いると，プログラマーは，いくつもの異なる形式のレコードを同じファイルに入れておくよう，指定することが可能になる。たとえば，

```
DCL 1 STR1 BASED(P),
      2 CODE CHAR(1),
      2 X CHAR(30),
    1 STR2 BASED(Q),
      2 CODE CHAR(1),
      2 X(8) FIXED BIN;
        ⋮
READ FILE(IN) SET(P);
IF STR1.CODE='2' THEN DO;
                       Q=P;
                       I=Q->X(1);
        ⋮
```

このプログラム・セグメントでは，各基底付構造はそれぞれ，その構造を識別するための要素CODE をもっている。レコードが読み込まれ，その位置がPの中にセットされる。CODE の値に応じて，レコードをSTR2と解釈できるように，Pの値がQに割り当てられる。

長さの変わる要素ストリングを，位置指定モードを用いて転送する場合は，ENVIRONMENT属性のSCALARVARYINGオプションをそのファイルに対して指定しなければならない(第12章「レコード方式の転送」をみていただきたい)。レコードには，長さ2バイトの接頭語が含まれる。

自己定義のデータ（REFERオプション）

自己定義のレコードとは，その中に含まれるフィールドに関する情報，たとえばストリングの長さなど，自分の中に含んでいるレコードである。基底付構造は，このようなデータを処理することができるように宣言することができる。ストリングの長さ，配列の限界，およびエリアのサイズはすべて，構造の中で宣言されている変数によって定義することができる。その構造が割り振られるとき（ALLOCATEステートメント，またはLOCATE ステートメントのどちらかによって）に，長さ，限界，またはサイズを定義する変数に，式の値が割り当てられる。これ以外にそれらの構造が参照された場合は，定義された変数の値が用いられる。

REFERオプションは，基底付構造の宣言の中で用いられるもので，その構造の割振りが行なわれるときに，その構造内の変数に式の値を割り当てること，またその式の値は，その構造内の別の変数の長さ，限界，またはサイズを表わしていることを指定するものである。REFERオプションの一般形式は，次の通りである。

```
要素式 REFER
（要素変数）
```

要素式の値は，整数に変換できるものでなければならない。式の中でオペランドとして用いられる変数は，REFERオプションを含んでいる構造に属するものであってはならない。

要素変数（REFERオプションの目的語ともいう）は，宣言されている構造のメンバーの名前でなければならない。これは，ロケーター修飾や添字のついているものであってはならないし，また，それが定義しているメンバーの前に~~あってはならない~~あるべきである。たとえば，

```
DECLARE 1 STR BASED(P),
          2 X FIXED BINARY,
          2 Y (L REFER (X)),
        L FIXED BINARY INITIAL(1000);
```

この宣言は，基底付構造STRが配列Yと要素Xで構成されていることを指定している。STRが割り振られるとき，その上限は，Xに割り当てられるLの現在の値にセットされる。このほかにYを参照したときは（PをセットするREADステートメントなど），その限界の値は，Xからとられる。

次にあげる制約が1つでも満足されていれば，構造の宣言の中でいくつでもREFERオプションを使用することができる。

1. REFERオプションのすべての目的語が論理レベル2で宣言されている（つまり，小構造の中で宣言されていない）。たとえば，

```
DECLARE 1 STR BASED,
          2 (M,N),
          2 ARR(I REFER (M),
            J REFER(N)),
          2 X;
```

この構造が割り振られるとき，IとJに割り当てられた値によって，2次元の配列ARRの限界がセットされる。

2. 構造のメンバーとメンバーの間にパディングが起らないように，構造が宣言されている。K節「データ・マップ」のところで，構造のマップを作るときの規則について述べる。たとえば，

```
DECLARE 1 STR UNALIGEND BASED (P),
          2 B FIXED BINARY,
          2 C,
            3 D FLOAT DECIMAL,
            3 E (I REFER (D))
              CHAR(J REFER (B)),
          2 G FIXED DECIMAL;
```

この構造はUNALIGNED属性をもっているので，どの項目もバイト境界に合ってさえいればよい。したがって，BとD（REFERの目的語）の値の如何に関わりなく，パディングは起らない。Dは，小構造の中で宣言されていることに注意。

3. 構造の宣言の中でREFERオプションが1度だけ使用されている場合は，次の条件に適っていれば，制約1と2を無視することができる。

   a. ストリングの長さ，またはエリアのサイズに関して，構造の最後の要素にこのオプションが適用されている。

   b. 配列の限界に関して，構造の最後の要素，または最後の要素が含まれている小構造のどちらかにこのオプションが適用されている。この配列の限界は，先頭の次元の上限でなければならない。たとえば，

```
DCL 1 STR BASED (P),
      2 X FIXED BINARY,
      2 Y,
        3 Z FLOAT DECIMAL,
        3 M FIXED DECIMAL,
      2 D (L REFER (M)),
        3 E (50),
        3 F (20);
```

この宣言は，制約1または2の条件を満足していない。つまり，REFERの目的語Mは，小構造の中で宣言されているし，パディングも起っている。しかし，このREFERオプションは，最後の要素が含まれている小構造に適用されているので，制約3が満足されている。

一段高いレベルから，配列の先頭の次元を受けつぐことができる点に注意していただきたい。たとえば，上の例で，STR(4)と宣言したとすれば，Dの先頭の次元はSTR(4)から受けつがれることになり，Dの中でREFERオプションを使うことができなくなる。

REFERオプションの目的語の値が，プログラムの間に変化する場合は，

1. 目的語が，それが割り振られたときにもっていた値に復元されるまで，構造を開放してはならない。

2. 目的語が，それが割り振られたときの値よりも大きい値をもっている間は，構造を書き出してはならない。

3. 目的語が，それが割り振られたときにもっていた値と同じか，それ以下の値をもっているときは，構造を書き出してもよい。実際に書かれる要素の数，ストリングの長さ，またはエリアのサイズは，目的語の現在の値によって表わされるものになる。たとえば，

```
DCL 1 REC BASED (P),
      2 N,
      2 A (M REFER(N)),
    M INITIAL (100);
          :
ALLOCATE REC;
N = 86;
WRITE FILE (X) PROM (REC);
```

この例では，RECの86個の要素が書かれる。Nは，割り振られたときにもっていた値（つまり，100）に復元しなければならないので，この時点でRECを解放するとエラーになる。Nに100より大きい値が割り当てられた場合は，WRITEステートメントに出会ったとき，エラーが起こる。

リスト処理

リスト処理は，データの集まりを操作する助けとなるいくつかの扱法につけられた名前である。PL/Iの配列や構造もデータの集まりを操作するのに用いられるものではあるが，リスト処理の技法は，プログラムの実行中にデータの集まりを作成したり，修正したりすることができるという点で，さらに柔軟性をもつ技法である。ここでは，これらの技法の例を詳述するのが目的ではなく，単に，このタイプの処理の基礎となるものとして，基底付変数とロケーター変数が如何に役に立つかを示すのが，その目的である。

リストの中にある別のメンバーの位置を識別するポインターを，各メンバーの中に最低1個はもっているリストは，連鎖された（または縫い合わされた）リストと呼ばれる。このようなリストを作るために，ALLOCATEおよびFREEステートメントが用いられる。

基底付変数に対するALLOCATEステートメント

このステートメントの形式は，次の通りである。

```
ALLOCATE 基底付変数
〔IN(エリア変数)〕
〔SET(ロケーター変数)〕
〔,基底付変数〕
〔IN(エリア変数)〕
〔SET(ロケーター変数)〕…;
```

基底付変数のデータ・タイプまたは構成は，どのようなものであってもよい。基底付変数が，ポインター変数と結びつけずに宣言された場合，または，基底付変数といっしょに宣言されたポインターを固定しておき，割り振られている基底付変数の世代を指し示すためには別のポインターをセットする，という必要がある場合には，SETオプションが必要である。

基底付変数に対するFREEステートメント

ステートメントの形式は，次の通りである。

```
FREE〔ロケーター修飾子 ->〕
基底付変数〔IN(エリア変数)〕
〔,〔ロケーター修飾子 ->〕
基底付変数〔IN(エリア変数)〕〕…;
```

ステートメントの中でポインター修飾子を指定することによって，基底付変数の特定の世代が解放される。修飾子が省略されているときは，その宣言の中で基底付変数と結びつけられているポインター変数が使用される。このような場合に，基底付変数にポインター変数が結びつけられていなかったときは，エラーになる。

基底付変数の複数個の世代

適当なポインター変数を指定することによって，基底付変数の現在の世代のすべてを参照することができる。リスト処理では，多くの世代をもついくつもの基底付変数を，1つのリストに組み入れることが可能である。リスト内のいくつかのメンバーは，あるメンバーの中にある，別のメンバーの位置を識別するポインターによって，互いに連鎖される。基底付変数の割振りでは，その変数を主記憶域内のどこに割り振るかということを指定することはできないことに注意していただきたい。

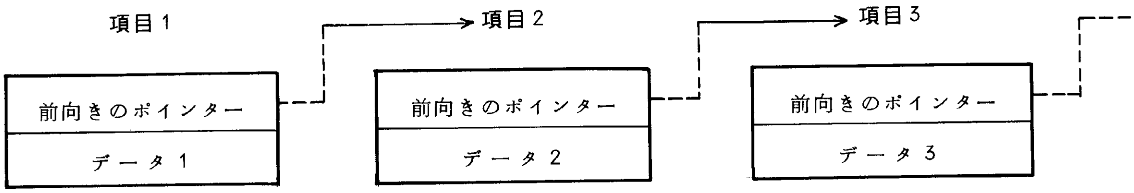


図 8. 1　一方向の連鎖の例

実際に，項目のチェイン（連鎖）は，主記憶域全体に散在することもあるが，各々のポインターをアクセスすることによって，次のメンバーが見つけられる。リスト内のメンバーは，通常，ポインター変数を含む構造である。たとえば，

```
      DCL  1  STR  BASED(H),
               2  P  POINTER,
               2  DATA,
              T  POINTER;
               :
               ALLOCATE  STR;
               T=H;
               :
   NEXT:ALLOCATE  STR  SET(T->P);
               T=T->P;
               :
               GO  TO  NEXT;
```

このプログラム・セグメントでは，構造のリストが作られる。これらの構造は，ＳＴＲの世代であり，各世代の中にあるポインター変数Ｐによって連係される。リストの作成中は，独立したポインター変数Ｔによって，前の世代が識別される。最初のALLOCATEステートメントは，それを識別するためにポインターＨをセットするものである。最終的に，ポインターＨは，リストの始まり（つまり先頭）を識別する。２番目の ALLOCATE ステートメントは，この新しい世代の位置を識別するために，前の世代の中のポインターＰをセットする。T=T->P；という割当てステートメントは，新しい世代の位置を識別するために，ポインターＴを更新するものである。

図 8. 1 は，この連鎖を図で表わしたものである。

各世代の中にあるＰの値が，別のポインター変数（たとえば，Ｐの上に定義されているポインター変数）の中に保存されていない限り，ＳＴＲの世代は，リストが作成された順序でしかアクセスすることはできない。上の例では，次のようなステートメントを使用して，各世代を順々にアクセスすることができる。

```
        T=H;
  NXT:T->DATA=X;
        :
        T=T->P;
        GO  TO  NXT;
```

NULL組込み関数

上で述べた方法でリストを作成するときは，リストの終りを表示する必要がある。NULL 組込み関数は，記憶域内の位置を識別できないポインター値を返す。したがって，リスト内の最後の世代の中のポインターを，NULL の値にセットすることによって，リストの終りの確実な表示をすることができる。たとえば，

```
          T=H;
  NXT:    IF  T¬=NULL  THEN
                         DO;
                         T->DATA=X;
                         :
                         T=T->P;
                         GO  TO  NXT;
                         END;
```

このプログラム・セグメントは，リストを走査するために，前の例のかわりに使用することができる。ＳＴＲの最後の世代の中のポインターＰは，NULL の値にセットされているものとする。

一般に，NULL 組込み関数の値は，ポインター（またはオフセット）変数が，記憶域内の位置を識別してはならないときに用いられる。ポインターにナルの値を与えるための唯一の方法は，NULL 組込み関数の値を割り当てる方法である（特別な場合が１つだけある。これについては，Ｇ節「組込み関数と擬似変数」のところで ADDR 組込み関数の項をみていただきたい）。基底付変数の世代をもはや識別しなくなったポインター変数の値（たとえば，基底付変数が解放されてしまったあとなど）については，わからない。

リストのタイプ

これまでに上げた例は，単純なリスト処理の扱法(一方向のリストの作成)を示すものであった。この構造に別のポインター変数を加えることによって，もっと複雑なリストを作ることができる。第2のポインターをこれに加えると，前の世代を指し示すことも可能になる。このようにすると，リストは両方向のものになり，適当なポインター値を使用すれば，リスト内のどの項目からも，前の項目と次の項目をアクセスすることができるようになる。最後のポインター値をNULLの値にセットするかわりに，リストの最初の項目を指すようセットして，輪または循環のリストを作ることができる。

1つの基底付変数の世代だけでリストを構成する必要はない。2つ以上の適当なポインター値をセットすることによって，いくつかの異なる基底付構造の世代をリストに組み入れることができる。ポインターの値を操作することによって，項目をリストに追加したり，削除したりすることができる。ポインターをうまく操作することによってリストの構造を作り直すことができるので，リスト内のデータの処理は，簡単になる。

主記憶域内でのデータの移動の量を減らすことによって，プログラマーは，通常，処理時間を大巾に節減することができる。しかし，ポインターは各々4バイトの記憶域を必要とし，さらに割り振られた基底付変数はいずれも，たとえばそれが長さ1バイトのビット・ストリングであっても，最低8バイトの記憶域を必要とすることに注意していただきたい。

エリア

基底付変数が割り振られるときは，使用できるところであればどこからでも記憶域がとられる。したがって，割り振られた基底付変数のリストは，主記憶域全体にわたって広く分散していることもあり得る。リスト上での内部的な操作の場合は，項目はポインターを使って容易にアクセスすることができるので，問題はない。しかし，リストをデータ・セットに転送する場合は，諸項目は1つにまとめておかなければならない。エリア変数の中に割り振られている項目は，すでに1つにまとめられており，それらは別々の存在のままでありながら，1つの単位として転送したり，割り当てたりすることができる。

エリア変数の先頭から相対的にみて，エリア変数内における基底付変数の位置を識別することができれば，都合がよい。オフセット変数は，このような目的で定義される。ポインター変数を使用すると，エリア変数が主記憶域に送り返されるとき，それらは有効でなくなる可能性が強い。

エリア変数

AREA属性は，基底付変数の割振りのために確保される記憶域のエリアを定義するものである。エリア変数の宣言は，次のような形式をしている。

DCL 識別名 AREA〔(サイズ)〕;

確保する記憶域の量は，バイト単位で与える(つまり，「サイズ」の整数値)。サイズが与えられていないときは，省略時解釈によって1000バイトとみなされる。

エリアのサイズは，ストリングの長さや配列の限界と同じように調節可能であるため，式または星印によって(被制御エリアまたはパラメーターの場合)，あるいはREFERオプションによって(基底付エリアの場合)，指定することができる。エリアの大きさの最大値は，そのプログラムが使用できる主記憶域の量によってのみ制約される。

宣言されたサイズのほかに，さらに16バイトの制御情報(使用中の記憶域の量などの詳細が含まれている)が，確保されたサイズのエリアの前につけられる。

確保された記憶域のうち，実際に使用されている記憶域の量を，エリアの範囲という。基底付変数は，実行中いつでも，エリアの中で割り振ったり，解放したりすることができる。このことは，記憶域が使用されるとき，エリアの範囲が変わることを意味している。どのような基底付変数もエリアの中で割り振ることができるので，これらは，それぞれ別の量の記憶域を要求することも可能である。基底付変数が解放されると，それが入っていた記憶域は，他の割振りに使用できるものとしてマークがつけられる。実際には，このコンパイラーは，エリア内の使用可能な記憶域のチェインを作って維持している(チェインの先頭が，16バイトの制御情報の中に貯えられている)。記憶域の必要量の異なる基底付変数が割り振られたり，解放されたりするときに，割振りが使用可能なスペースに収容できなければ，エリアの中にギャップができてしまうことは避けられない。したがって，すでに解放されてはいるが，まだ意味をもっている割振りがエリアの範囲に入っていることもある。意味をもっている割振りとは，まだ解放されていないか，または解放されてはいるが，その後にまだ解放されていない割振りが少なくとも1つはある割振りである。エリアが，意味をもっている割振りをもっていないときは，範囲はゼロになる。

エリア変数に対しては，どのような演算子（比較演算子でさえも）も使うことはできない。

オフセット変数

オフセット変数は，エリア変数といっしょに排他的に用いられる特別な形式のポインターである。オフセット変数の値は，エリアの先頭から相対的にみた，エリア変数内における基底付変数の位置を表わしている。基底付変数は相対的に識別されるので，エリア変数が主記憶域の別の部分に割り当てられる場合も，オフセット値は無効にならない。オフセット変数を使用しても，エリア内でポインター変数を使用することが禁止されるわけではないことに注意していただきたい。オフセット変数は，次のようにして宣言される。

```
        DCL  識別名
    OFFSET〔（要素エリア変数）〕；
```

エリア変数とオフセット変数の結びつけは，特別な関係ではない。オフセット変数は，POINTER組込み関数を用いて，どのようなエリア変数とでも結びつけることができる（下の「ロケーター変換」の項を参照）。宣言の中でこのような結びつけをしておくと，オフセット変数を参照すると，結びつけられたエリア変数を暗に指すことになるので便利である。オフセットの宣言の中にエリア変数があると，エリア変数を文脈的に宣言したことになることに注意していただきたい。

ロケーター変換

オフセット変数が参照の中で用いられているときは，これは暗示的にポインター値に変換される（つまり，結びつけられたエリア変数のアドレス値が，オフセット値に加えられる）。オフセットからポインター値への明示的な変換は，POINTER 組込み関数を用いて行なわれる。たとえば，

```
    DCL P POINTER,O OFFSET(A),B AREA;
                 ⋮
    P = POINTER(O,B);
```

このステートメントは，ポインター値をPに割り当てるもので，エリアBの中にあるオフセットOによって識別される基底付変数の位置が得られる。このエリア変数は，オフセット変数と結びつけられたものとは違うので，プログラマーは，そのオフセット値が，別のエリアについても有効であるか確認しなければならない。たとえば，この関数を呼び出す前にエリアAがエリアBに割り当てられた場合にも有効である。

OFFSET組込み関数は，POINTER組込み関数を補うものであり，与えられたポインターとエリアから得られたオフセット値を返す働きをする。与えられたポインター値は，与えられたエリア内における基底付変数の位置を識別するものでなければならない。

実際問題として，変換はほとんどの場合，暗示的に行なわれるので，これらの関数を使うことはあまりない。

オフセット式

オフセットは，暗示的にポインター値に変換されるので，オフセット式は，ポインター式と交換できるように使用することができる。オフセット式は，ロケーター修飾子として使用したり，基底付変数の宣言との結びつけ，比較演算の中，あるいはプロシージャー関数の引数として，使用することができる。しかし，READ または LOCATE ステートメントのSETオプションの中で指定することはできないので，注意していただきたい。

INオプションをもつALLOCATEステートメント

オフセット値は，最初は，ポインター値を変換することによって，あるいは ALLOCATE ステートメントのSETオプションを使うことによって，得ることができる。この形式の ALLOCATE ステートメントは，次の通りである。

```
    ALLOCATE  基底付変数
              〔IN（要素エリア変数）〕
              〔SET（ロケーター変数）〕；
```

このステートメントは，指定されたエリアの中で基底付変数に対して記憶域を割り振るものである。すると，この変数はエリアの先頭からの相対的なオフセットをもつことになり，このオフセット値が，SETオプションで指定されたロケーター変数に割り当てられる。このロケーター変数がポインター・タイプの場合は，変換が行なわれる。

INまたはSETオプションのどちらか一方または両方を暗示することができる。たとえば，

```
    DCL  X  BASED(O),
         Y  BASED(P),
         A  AREA,
         O  OFFSET(A);
         ⋮
         ALLOCATE  X;
         ALLOCATE  Y  IN(A);
```

エリアAとオフセットOの記憶域クラスは，省略時解釈によって，AUTOMATICになる。最初の ALLOCATE ステートメントは，次のものと同じである。

```
    ALLOCATE  X  IN(A)  SET(O);
```

2番目の ALLOCATE ステートメントは，次のものと同じである。

```
ALLOCATE Y IN(A) SET(P);
```

プログラマーは，すべての暗示が解決できるものであるか確かめなければならない。たとえば，オフセットOが基底付変数と結びつけられていなかったとすれば，SETオプションが必要になる。

INおよびSETオプションが暗示でなく指定されている場合は，INオプションの中にあるエリア以外のエリアと結びつけて，あるいはエリアを何も結びつけずに，宣言されたオフセット変数を使用することができる。次の例の ALLOCATE ステートメントは両方とも正しい。

```
DCL T1 OFFSET (A1),
    T2 OFFSET,
    A2 AREA;
       :
ALLOCATE X IN (A2) SET (T1);
ALLOCATE Y IN (A2) SET (T2);
```

オフセット変数T1とT2は，それぞれ，変数XとYのオフセットの値をエリアA2の中にもっている。

次の例は，オフセット変数を用いて，エリア変数の中にリストを作る方法を示したものである。この例は，この章の前の方の「基底付変数の複数個の世代」のところであげた例を書き直したものである。

```
DCL A AREA,
    (T,H) OFFSET(A),
    1 STR BASED(H),
      2 P OFFSET(A),
      2 DATA;
    :
    ALLOCATE STR IN(A);
    T=H;
    :
NEXT:ALLOCATE STR SET(T->P);
    T=T->P;
    :
    GO TO NEXT;
```

これと同じ ALLOCATE ステートメントを使って，複数個の割振りを行なうことができる。たとえば，

```
ALLOCATE X, Y IN(A);
```

基底付変数と被制御変数の両方を，同じステートメントの中で割り振ることができる。

INオプションをもつFREEステートメント

エリア変数の中で割り振られた基底付変数は，INオプションによってエリア変数を指定することにより，解放することができる。

```
FREE 基底付変数
     〔IN(要素エリア変数)〕;
```

同じFREE ステートメントを用いて，基底付変数と被制御変数の両方をいくつかまとめて解放することができる。エリア変数の中にある変数の現在の割振りのすべてを解放するときは，EMPTY組込み関数を使うのが，最も便利である。

EMPTY組込み関数

エリア変数が割り振られるとき，エリア変数は自動的に空白の状態，すなわち，エリアの範囲がゼロ，になる。EMPTY組込み関数の値をエリア変数に割り当てることによって，その変数の中のすべての割振りを解放することができる。この関数参照では引数は不要であるが，名前が BUILTIN と宣言されていない場合には，ナル引数リストを与えなければならない。たとえば，

```
DECLARE A AREA,
        I BASED (P),
        J BASED (Q);
          :
ALLOCATE I IN(A), J IN (A);
          :
A= EMPTY ();
/*EQUIVALENT TO:
  FREE I IN (A), J IN (A); */
```

エリア変数自体は解放されず，その記憶域は，後で基底付変数に割り振るためにそのままにしておかれることに注意していただきたい。

エリアの割当て

割当てステートメントを用いて，エリア式の値を1つまたは2つ以上のエリア変数に割り当てることができる。エリアからエリアへの割当ては，原エリアのすべてのオフセットが目標エリアでも有効になるようにして，目標エリアのすべての割振りを解放し，そのあとで原エリアの範囲を目標エリアに割り当てるという効果をもっている。たとえば，

```
DECLARE X BASED (O(1)),
        O(2) OFFSET (A),
        (A,B) AREA;
          ⋮
        ALLOCATE X IN (A);
        X = 1;
        ALLOCATE X IN (A) SET (O(2);
        O(2) -> X = 2;
        B = A;
```

このプログラム・セグメントが与えられているときにPOINTER組込み関数を使用すると，POINTER(O(2),B)->X およびO(2)->X という参照は，それぞれエリアBとAに割り振られたものと同じ値を表わすことになる。

割振りが何も含まれていない原エリアが目標エリアに割り当てられる場合は，その効果は，目標エリア内のすべての割振りを解放するだけである。

エリアの割当ての用法として，そのもとになるエリアの限界を越えて基底付変数のリストを拡張することができる。基底付変数をエリアの中に割り振るとき，それを収容できるだけの解放された記憶域が充分にないと，ARER条件が起こる（以下参照）。この条件に対するオン・ユニットは，不適当なエリアへの参照を修飾しているポインターの値を変えて，別のエリアを指すようにすることもできる。そして，オン・ユニットから戻ったときに，新しいエリアの中で再び割振りを行なってみる。かわりに，オン・ユニットでエリアを書き出し，エリアをEMPTYにリセットしてもよい。

AREAオン条件

次のような状況のいずれかでは，AREA条件が起こる。

1. その割振りを行なうための解放された記憶域が充分でないエリアの中に，基底付変数を割り振ろうとしたとき

2. エリアの割当てを行なおうとしたときに，目標エリアが小さすぎて原エリアの範囲を収容できないとき

3. SIGNAL AREAステートメントが実行されたとき

割振り，割当て，またはSIGNAL ステートメントによって，この条件が起ったかどうか調べるために，ONCODE組込み関数を使用することができる。オン・ユニットから正常に戻るときにとられる処置は，次の通りである。

1. 割振りによってこの条件が起ったときは，割振りがもう一度行なわれる。不適当なエリアへの参照を修飾しているポインターの値が別のエリアを指すよう，オン・ユニットによって変えられた場合は，新しいエリアの中で割振りが再び行なわれる。オン・ユニットがこの誤りをうまく訂正できないときは，ループが起こる可能性もあることに注意していただきたい。

2. エリアの割当て，またはSIGNAL ステートメントによってこの条件が起ったときは，割込みの地点から実行が続けられる。

オン・ユニットが指定されていないときは，システムが注釈を出し，ERROR条件を引き起す。

エリアの入力と出力

エリア機能は，基底付変数の完全なリストを1単位として，RECORDファイルとの間で入力と出力を容易に行なうことができるよう設計されたものである。出力の場合には，16バイトの制御情報といっしょにエリアの範囲だけが転送される（この範囲には，解放されてはいるがまだ意味をもっている割振りが含まれているが）。したがって，エリアの使用されていない部分は，データ・セット上にスペースをとらない。エリアの範囲は変わる可能性があるので，V一形式またはU一形式レコードを使用しなければならない。レコードに必要な最大の長さは，エリアの長さによって左右される（つまり，エリアのサイズ＋16）。

複数個のロケーター修飾

ロケーター修飾は，基底付変数の特別な世代を識別するために，基底付変数と1つまたは2つ以上のロケーター値を結びつけることである。基底付変数への参照は，次のようにして明示的に修飾することができる。

要素ロケーター式 ->
〔基底付ロケーター変数 ->〕…
基底付変数

ロケーター修飾の用法について，いくつかの一般規則を記述することができる。

1. ロケーター修飾は，結びつけられている参照が適用される基底付変数の世代を表示するために用いられる。

2. オフセット式またはオフセット変数がロケーター修飾として用いられる場合は，基底付変数が参照されるたびに，その値は，ポインター値に暗示的に変換される。

3. 1つの参照の中で2つ以上のロケーター修飾子が用いられている場合は，最初の（つまり左端の）修飾子だけが，関数参照であってよい。その他のロケーター修飾子はすべて，それ自身，基底付変数でなければならない。しかし，入口変数は基底付であってよく，またロケーター値を戻す関数を表わすものであってよいことに注意していただきたい。

4. 2つ以上のロケーター修飾を使用しているときは，これらは左から右に評価される。

基底付変数への参照は，暗示的に修飾されたものであってもよい。暗示的に修飾されている基底付変数の世代を決めるために用いられるロケーター値は，基底付変数といっしょに宣言されたものである。基底付変数といっしょに宣言されたロケーターは基底付であってもよいので，ロケーター修飾子の連鎖を暗示することもできる。たとえば，

```
DECLARE (P(10),Q) POINTER,
        R POINTER BASED (Q),
        V BASED (P(3)),
        W BASED (R),
        Y BASED;
```

上の宣言が与えられているとすれば，次にあげる参照は有効である。

1. P(3) -> V
2. V
3. Q -> R -> W
4. R -> W
5. W

参照1と2は，参照3，4および5と同じである。Yは，ロケーター変数がそれと結びつけられるまで，参照することができないことに注意していただきたい。

ロケーター修飾のレベル

基底付変数を修飾するポインターは，1つのレベルのロケーター修飾を表わしており，オフセットは，エリアの中で暗示的に修飾されるので，2つのレベルを表わしている。レベルの数は，添字のついたロケーター，および／または構造の要素によって影響されることはない。1つの参照の中で許されるロケーター修飾のレベルの最大の数は，10である。たとえば，

```
DECLARE X BASED (P),
        P POINTER BASED (Q),
        Q OFFSET (A);
```

上の宣言が与えられているとすれば，X，P->X，およびQ->P->Xという参照はすべて，3つのレベルのロケーター修飾を表わしている。

第9章： サブルーチンと関数

PL/Iのもつブロック構造の概念は，サブルーチンおよびプログラマー定義の関数を使用することを可能にしている。サブルーチンおよび関数は，次のようなことを行なうことのできるステートメントのグループである。

1. 頻繁に使用される同じ処理を行なうために，プログラムの中のいろいろな地点から呼び出す。
2. さまざまな地点の呼出しから渡されたデータを処理する。
3. 呼出し地点のすぐ後の地点に制御を戻す。さらに関数の場合には，関数を実行した結果得られた値も戻す。

サブルーチンおよび関数は，呼出し側ブロックの内部にあっても，外部にあってもよい。組込み関数はつねに，PL/Iのシステム環境の中に永久的に維持される外部プロシージャーであり，PL/I言語に不可欠な一部分をなしている。

サブルーチンおよび関数の用法に関してこの章であげてある規則は，そのサブルーチンまたは関数が外部または内部プロシージャーのどちらであるかによって左右される。これは，呼び出されたプロシージャーが呼出し側プロシージャーの内部にあるときは，2つのプロシージャーの間の関係を，そのプロシージャー自体からコンパイラーが判定することができるためである。呼び出されたプロシージャーが外部にあるときは，呼出し側プロシージャーの中で，その関係を明示的に与えなければならない。したがって，コンパイラーが所要の目的プログラムを作成することができるようにするために，呼出し側プロシージャーの中で外部サブルーチンまたはプロシージャーに関するさらにくわしい情報を与える必要がある。

サブルーチンは，CALLステートメント，またはINITIAL属性のCALLオプションによって呼び出されるプロシージャーである。

関数は，プログラマー定義の場合も，組込み関数の場合も，式の中に"関数参照"があると，呼び出される。関数参照は，関数の入口名を表わす入口式である。（入口名は，プロシージャーの特定の入口点を表わす識別名である。）

PL/Iにおけるサブルーチンと関数の決定的な相違は，サブルーチンは呼出し地点にデータの値を返さないのに対して，関数参照は通常，関数参照が記入されている式を評価して，関数参照と置き換えるために値を返す，という点である。

サブルーチンと関数はどちらも，呼出し側ブロックの中で知られているデータを使用することができる。データを使用できるようにするための方法は，2通りある。

1. 呼出し側ブロックと呼び出されたプロシージャーの両方に知られている名前によって表わされるデータ。名前が知られている場所を決めるための規則については，第7章「名前の認定」をみていただきたい。
2. 引数とパラメーター： CALLステートメントまたはCALLオプション，あるいは関数参照につけられた引数リストの中に引数を書くことによって，呼出し側ブロックからの値を呼び出されたプロシージャーに渡すことができる。これらの値は，呼び出されたプロシージャーの中では，パラメーターを用いて使用することができる。

   パラメーターは，呼び出された入口点のパラメーター・リストの中にある識別名である。引数とパラメーターの数は，同じでなければならない。~~入口点として許される数は，最高64まで~~である。パラメーターには，記憶域は結びつけられない。これは，呼び出されたプロシージャーが，呼出し側プロシージャーの中に割り振られた記憶域をアクセスできるようにするための単なる手段である。プロシージャーの中でパラメーターを参照すると，実際には対応する引数を参照することになる。パラメーターの値を変更すると，引数の値も変更される。しかし，ある特定の環境の場合は，見かけ引数が作られ，もとになる引数の値は変更されない。これは，次のような場合である。

   a. 引数の属性が，対応するパラメーターの属性と違っているとき。もとになる引数の値は，変換され，見かけ引数に割り当てられる。

   b. 1つの値だけが引数として渡されたとき，たとえば，引数が定数であるときなど。

   c. 引数がiSUB定義の配列であるとき。

このような場合は，パラメーターを参照すると，実質的には見かけ引数が参照される。見かけ引数とパラメーターは，最初は同じ値をもっているが，のちにこのパラメーターに変更を加えても，もとの引数の値は影響されない。

見かけ引数に対する記憶域は，呼出し側プロシージャーに属する記憶域の中にある。

サブルーチンと関数の入口点

サブルーチンまたは関数プロシージャーは，１つまたは２つ以上の入口点をもつことができる。

PROCEDUREステートメント：　プロシージャーへの主要入口点は，PROCEDUREステートメントによって設定される。

ENTRYステートメント：　プロシージャーへの副次入口点は，ENTRYステートメントによって設定される。

PROCEDUREステートメントと補助的なENTRYステートメントはそれぞれ，自分自身のパラメーターと，関数参照の場合には返される値の属性を指定することができる。しかし，PROCEDUREステートメントの位置からブロックに入るときに設定される環境は，同じブロックが副次入口点で呼び出されるときに設定される環境と全く同じである。各入口点は，それに結びつけられた入口名をもっている。PL／Iプロシージャーへの外部入口点に対する名前の長さは，７桁に限られている。

入口名は，PROCEDUREまたはENTRYステートメントの接頭語としてつけておくことによって，内部プロシージャーに対する入口定数として呼出し側ブロックの中で明示的に宣言される。DECLAREステートメントの中で内部入口名として宣言するのは，誤まりである。外部入口名は，ENTRY属性をもつ入口定数として明示的に宣言しなければならない。入口変数は，それに割り当てられる入口定数を表わし，ENTRYおよびVARIABLE属性をもっている識別名である。入口変数を参照すると，最後に割り当てられた入口定数値が参照される。

ENTRY属性の用法

ENTRY属性の一般形式は次の通りである。

```
識別名　ENTRY
　　　　〔（パラメーター記述リスト）〕
　　　　〔VARIABLE〕
　　　　〔RETURNS　（属性リスト）〕
　　　　〔OPTIONS（オプション・リスト）〕
```

パラメーター記述子リストは，識別名によって表わされる入口点と結びつけられるパラメーターの属性を指定するために用いられるものである。詳細については，この章の「パラメーター記述リスト」のところで説明してある。

関数プロシージャーから返される値の属性を指定するために，RETURNS属性を与えることができる。

入口点が，外部関数の中，またはCOBOLまたはFORTRANコンパイラーによってコンパイルされたサブルーチンの中にある場合は，OPTIONS属性が必要である。詳細については，第１７章「言語間の連絡機能」のところで説明する。

関数またはサブルーチンが外部のものである場合は，入口宣言を与えなければならない。内部サブルーチンや内部関数に対して入口宣言をするのは誤りである。PROCEDUREまたはENTRYステートメントの中に名前を記入すれば，この場合，外部宣言をしたのと同じことになる。

入口宣言の中にあるパラメーター記述子リストには，引数の変換と見かけ引数の作成を制御するためのパラメーターの属性を入れる。コンパイラーが正しい見かけ引数を作ることができるように，パラメーター記述子には，パラメーターの属性に関する正確な情報を入れておかなければならない。外部入口宣言の中にパラメーター記述子リストがない場合は，コンパイラーは，引数の属性は対応するパラメーターの属性と一致しているものとみなさなければならない。変換は行なわれない。

サブルーチンと関数の出口点

呼出し地点のすぐ後の地点に制御を戻すために，RETURNステートメントが用いられる。どこか別の地点に制御を移すときは，GO TOステートメントが用いられる。ENDステートメントを用いて，RETURNステートメントと同じようにサブルーチン・プロシージャーから制御を返すこともできる。関数プロシージャーの場合は，関数参照が出てくる式の中で，関数参照に与えられる値をもつ要素式を指定しなければならない。

RETURNS属性とRETURNSオプション

RETURNS属性は，関数プロシージャーから受けとる値の属性を，呼出し側ブロックに対して指定するものである。RETURNSオプションは，返す値がもっているべき属性を，関数プロシージャーに対して指定するものである。該当する値がこれらの属性をもっ

ていない場合は，適当な変換を行なってから，関数は，制御を引き渡し，値を返すようにする。

RETURNSオプションが指定されていない場合は，入口点名の頭文字にしたがって，省略時解釈により，返される値の属性が想定される。標準の省略時解釈は，頭文字が（I：N）の範囲の場合はREAL FIXED BINARY（15，0）であり，（A：H）および（O：Z）の範囲と$，#，@の文字の場合は，REAL FLOAT DECIMAL（6）である。

内部入口名に対してRETURNS属性を指定してはならないが，これは，返される値の属性は，コンパイラーが関数プロシージャー自体から決めるためである。外部入口名または入口変数に対してこの属性が指定されていない場合は，コンパイラーは，関数から返される値に対して省略時解釈の属性を想定する（入口点名から判定する）。したがって，外部関数プロシージャーが返す値の属性が，省略時解釈では正しく判定できないという状況の場合は，RETURNS属性とRETURNSオプションの両方を与えなければならない。関数から返される値は，このオプションの中で指定された属性をもっており，一方，呼出し側のプロシージャーはつねに，その値はRETURNS属性の中で指定されている属性をもっているものとみなすので，RETURNS属性の中の属性とRETURNSオプションの中の属性は一致していなければならない。

サブルーチン

サブルーチン・プロシージャーの用法に関連するPL/Iステートメントについて，以下に説明する。

サブルーチンは，呼出しをするCALLステートメントの中でそれに渡される引数を通常必要とするプロシージャーである。これは，外部または内部プロシージャーのどちらであってもよい。このようなプロシージャーを参照することは，サブルーチン参照という。CALLステートメントまたはINITIAL属性のCALLオプション内におけるサブルーチン参照の一般形式は，次の通りである。

CALL　入口式
〔（引数〔，引数〕…）〕；

サブルーチンが呼び出されたときはいつでも，呼出しステートメントの引数が，入口点のパラメーターと結びつけられ，そのあとで制御がその入口点に渡される。このようにして，サブルーチンは活用化され，サブルーチン・プロシージャーの実行が始まる。

サブルーチンが終了すると，制御は通常，呼出し側ブロックに戻る。サブルーチンが終了するのは，そのサブルーチンの中に対応するオン・ユニットがないような条件を引き起すステートメントが与えられたとき，または次にあげるステートメントのいずれかが与えられたときである。

ENDステートメント：　サブルーチンの最後のENDステートメントに制御が達したとき。このステートメントが実行されると，そのサブルーチンを呼び出したCALLステートメントに制御が戻される。

RETURNステートメント：　サブルーチン内のRETURNステートメントに制御が達したとき，これは，ENDステートメントによる場合と，同じ正常な戻りである。

GO TOステートメント：　サブルーチンの外に制御を移すGO TOステートメントに制御が達したとき，（そのサブルーチンが，INITIAL属性のCALLオプションによって呼び出された場合には，これは許されない。）このGO TOステートメントは，その外側のブロックの中のラベルを指定してもよいし（このラベルはそのサブルーチンの中で知られているものでなければならない），そのサブルーチンに渡されたラベル引数と結びつけられたパラメーターを指定してもよい。これは，サブルーチンの終了の方法として有効ではあるが，ENDまたはRETURNステートメントによる場合のように，正常な戻りではない。

STOPステートメント：　サブルーチンの中でSTOPステートメントに出合うと，そのサブルーチン，およびそれを呼び出したプロシージャーと結びつけられるプログラム全体の実行が，異常停止させられる。

サブルーチンの用法：　次にあげる例は，サブルーチンが，それを呼び出すプロシージャーとどのように影響し合うかを示したものである。

例1：

```
PRMAIN:  PROCEDURE;
         DECLARE NAME CHARACTER (20),
           ITEM BIT(5), OUTSUB ENTRY;
            :
         CALL OUTSUB (NAME, ITEM);
            :
         END PRMAIN;
OUTSUB:  PROCEDURE (A,B);
         DECLARE A CHARACTER (20),
           B BIT(5);
            :
         PUT LIST (A,B);
            :
         END OUTSUB;
```

プロシージャーPRMAINの中で，NAMEは文字ストリングとして，ITEMはビット・ストリングとして宣言されている。PRMAINの中のCALLステートメントは，OUTSUBと呼ばれるプロシージャーを呼び出しており，このプロシージャ参照の中にある括弧付のリストには，OUTSUBに渡される2つの引数が含まれている。OUTSUBを定義するPROCEDUREステートメントは，AとBの2つのパラメーターを宣言している。OUTSUBが呼び出されると，NAMEはAと結びつけられ，ITEMはBと結びつけられる。OUTSUBの中でAを参照すると，それはつねにNAMEを参照したものとして扱われ，Bを参照すると，ITEMを参照したものとして扱われる。したがって，PUT LIST(A,B)ステートメントを与えると，NAMEとITEMの値が，標準システム出力ファイル，SYSPRINTに書き出される。NAMEとITEMの属性はそれぞれ，AとBの属性と一致しているので，PRMAINの中でのOUTSUBの宣言では，パラメーター記述子をENTRY属性と結びつける必要がないことに注意していただきたい。

PROCEDUREまたはENTRYステートメントのパラメーター・リストの中に名前を記入すると，その名前はパラメーターであると明示的に宣言されたことになる。しかし，省略時解釈を適用するのでない限り，その属性は，そのプロシージャー内のDECLAREステートメントの中で明示的に記述しなければならない。

引数とパラメーターを使用すれば，あるプロシージャーの中で名前を知られていないデータでも操作できるようにするために，プロシージャーを一般化することができることが，わかっていただけたことと思う。

例2：

```
A: PROCEDURE;
   DECLARE RATE FLOAT (10), TIME FLOAT(5),
          DISTANCE FLOAT(15),MASTER FILE;
          ⋮
   CALL READCM (RATE, TIME, DISTANCE,
          MASTER);
          ⋮
READCM: PROCEDURE (W,X,Y,Z);
        DECLARE W FLOAT(10), X FLOAT(5),
                Y FLOAT(15), Z FILE;
                ⋮
        GET FILE (Z) LIST (W,X,Y);
                ⋮
        Y = W*X;
        IF Y > 0 THEN RETURN;
             ELSE PUT LIST('ERROR READCM');
        END READCM;
   END A;
```

引数RATE，TIME，DISTANCE，およびMASTERは，パラメーターW，X，Y，およびZに渡される。したがって，このサブルーチンの中では，Wを参照することはRATEを参照するのと同じであり，XはTIMEを，YはDISTANCEを，ZはMASTERを，それぞれ参照するのと同じである。

関　数

サブルーチンはCALLステートメントかCALLオプションによって呼び出されるが，関数は，式の中に関数名（と関連する引数）を書くことによって呼び出される。このように関数名を書くことを，関数参照という。サブルーチンの場合と同様，関数も，渡された引数や，その他の知られているデータを操作することができる。しかし，サブルーチンとは異なり，関数は，1個の値だけを計算して，呼出し地点に制御といっしょにこの値を返すよう，書かれている。この1つの値は，入口タイプ以外のものならばどのようなタイプのものであってもよい。次にあげるプロシージャーの中には，関数参照の例が組み込まれている。

```
MAINP: PROCEDURE;
          ⋮
       GET LIST (A,B,C,Y);
          ⋮
       X = Y**3+SPROD (A,B,C);
```

上の例の中にある次の割当てステートメント

```
X = Y**3+SPROD(A,B,C);
```

には，SPRODと呼ばれる関数の参照が含まれている。関数名の後の括弧付のリストには，SPRODに渡される引数が含まれている。SPRODが以下のように定義されているとすれば，

```
SPROD: PROCEDURE (U,V,W);
          ⋮
       IF U > V + W
             THEN RETURN (0);
             ELSE RETURN (U*V*W);
          ⋮
       END SPROD;
```

SPRODがMAINPによって呼び出されると，引数A，B，CはそれぞれパラメーターU，V，Wと結びつけられる。引数やパラメーターに対して属性は明示宣言されていないので，各引数および各パラメーターには，省略時解釈の属性FLOAT DECIMAL(6)が適用される。

SPRODの実行中にIFステートメントに出会うと，テストが行なわれる。UがV+Wより大きいときは，THENクローズと結びつけられたステートメントが実行されるが，それ以外の場合は，ELSEクローズと結びつけられたステートメントが実行される。どちらの場合も，実行されるステートメントはRETURNステートメントである。

RETURNステートメント： RETURNステートメントは，関数を終了させるために普通用いられる方法であり，制御が呼出しプロシージャーに返される。関数の中でこのステートメントを使う場合と，サブルーチンの中で使う場合とでは，いくらか相違がある。関数の中では，これは制御を戻すばかりでなく，呼出し地点に値を返す働きもする。関数の中で使用する場合のRETURNステートメントの一般形式は，次の通りである。

RETURN （要素式）；

要素式の値は，呼出しプロシージャーの，呼出し地点のところに返される。したがって，上の例では，SPRODは，0かまたはU＊V＊Wによって表わされる値を制御といっしょに，MAINPの呼出し式に返す。返された値が，関数参照の値としてとられ，呼出し式の評価がつづけられる。

GO TOステートメント： GO TOステートメントを実行することによって，関数を終了させることもできる。この方法が用いられた場合は，関数を呼び出した式の評価は完了せず，指定されたステートメントに制御が移る。サブルーチンの場合と同様，GO TOステートメントの中で指定する移動の行先点は，ラベル引数と結びつけられたパラメーターであってもよい。たとえば，MAINPおよびSPRODが，次のように定義されていたとすれば，

```
MAINP:  PROCEDURE;
          ⋮
        GET LIST (A,B,C,Y);
        X = Y**3+SPROD(A,B,C,LAB1);
          ⋮
LAB1:   CALL ERRT;
          ⋮
        END MAINP;
SPROD:  PROCEDURE (U,V,W,Z);
        DECLARE Z LABEL;
          ⋮
        IF U > V + W
                THEN GO TO Z;
                ELSE RETURN (U*V*W);
          ⋮
        END SPROD;
```

MAINPの中で，LAB1は，CALL ERRTステートメントのラベルとして出てきているので，ステートメント・ラベル定数として明示宣言されている。SPRODが呼び出されると，LAB1はパラメーターZと結びつけられる。Zの属性は，LAB1の属性と一致していなければならないので，ZはLABEL属性をもつと宣言される。SPRODの中のIFステートメントが実行されると，テストが行なわれる。UがV+Wより大きいときは，THENクローズが実行されて，MAINPの中のLAB1というラベルのついたステートメントに制御が戻り，SPRODを呼び出した式の評価は中止される。UがV+Wより大きくないときは，ELSEクローズが実行されて，ごく普通の方法でMAINPに戻される。

ラベル引数およびラベル・パラメーターの用法に関するさらにくわしいことは，この章の「引数とパラメーターの関係」のところをみていただきたい。

注： 引数を必要としないような形で関数を定義することができる場合もある。このような場合は，外部関数名が式の中にあると，関数名が入口名であると明示的に宣言されているときだけ，関数参照と認められる。さらに詳しい点については，この章の「ENTRY属性」のところをみていただきたい。

返される値の属性

RETURNS属性： RETURNS属性は，外部入口名に対するDECLAREステートメントの中で指定する。これは，その関数から返される値の属性を呼出し例ブロックに対して指定するものである。また，これは，その名前に対して暗示的にENTRY属性も指定する。返される値の属性が，省略時解釈によって正しく決定できない場合は，外部関数の呼出しは，RETURNS属性をもつ入口名の宣言の有効範囲内になければならない。RETURNS属性の一般形式は，以下の通りである。

RETURNS （属性リスト）

RETURNS属性は，外部関数プロシージャーから返される値は，呼出しプロシージャーの中では，属性リストの中で与えられている属性をもっているように扱われることを指定するものである。

呼出し側ブロックでは，それに返される値がどのようなものであっても，変換は行なわないので，扱われるという語を使用した。
返される値は，RETURNSオプションから決められた属性をもっているので，RETURNS属性の中で与えられる属性は，対応するRETURNSオプションの中で与えられるデータ属性と一致していなければならない。

内部プロシージャーに対して RETURNS 属性を与えることはできない。返される値の属性は，入口点のところで RETURNS オプション（与えられていれば）から決められる。与えられていないときは，入口定数の識別名の場合と同様に，省略時解釈にしたがって決められる。

RETURNSオプション： RETURNSオプションは，関数プロシージャーのPROCEDUREまたはENTRY ステートメントの中で指定する。これは，関数から戻る前に，関数から返される値が変換される属性を指定するものである。

総称入口名と参照

総称入口名は，プロシージャー入口点の集まりを表わすものであり，そのメンバーはそれぞれ，総称参照，すなわち，実際の入口名のかわりに総称名を用いるプロシージャー参照によって，呼び出すことができる。呼び出されるメンバーは，総称参照の中で指定されている引数の数と属性にしたがって，決められる。呼び出されるメンバーは，その数と属性の両方が，引数と合致するような総称記述子リストをもつ最初のメンバーである。

総称名は，GENERIC属性をつけて宣言しなければならない。この属性の一般的な形式は，次の通りである。

```
総称名　GENERIC　(入口式
  WHEN　(総称記述子リスト)
  〔，入口式
  WHEN (総称記述子リスト)〕…)；
  なお，総称記述子リストは以下の通り；
  (〔記述子〔，記述子〕…〕)
```

入口式は，それぞれ，この集まりの中にあるプロシージャー入口点の1つ1つに対応している。入口式は，入口名を表わす入口名または入口式にすることができる。総称記述子リストの中の記述子はそれぞれ，引数に1つ1つ対応しており，対応する引数がもっているべき属性を，関連する入口名が選択できるような順番で，指定することができる。記述子が不要な場合は，省略することも，また，星印で表示することもできる。省略されている記述子が，唯一の記述子である場合は，星印の形式は重要である。たとえば，（，）は2つの記述子を表わしているのに対して，(*)は1個を表わす。引数がないことを表わす総称記述子リストは，次のような形式をとる。

```
……ENTRY1　WHEN(　)……
```

入口式は，総称宣言の中で指定されたものの中から，総称選択といわれる処理によって選ばれる。 総称選択は，関数参照または CALL ステートメントの中で指定された引数と，GENERIC宣言の中の各入口式で与えられた総称記述子リストの内容とくらべることによって，行なわれる。最初に，各総称記述子リストが，宣言の中に出てきた順にチェックされ，総称名への参照の中にある引数と同じ数の記述子が入っているかどうか調べられる。

引数と同じ数の記述子をもつ総称記述子リストが見つかると，各記述子は，対応する引数とくらべてテストされ，記述子の中で与えられたすべての属性が引数の属性であるかどうか調べられる。たとえば，総称記述子リストに次のものが含まれており，

```
………(FLOAT, FIXED)
```

しかも，対応する2つの引数が，明示的，文脈的，暗示的，または省略時解釈のいずれかによって，DECIMAL FLOAT (6) およびBINARY FIXED (15，0)などという属性をもっている場合は，総称記述子リストの中の各属性は，対応する引数の属性と同じであり，この選択はうまくいったわけである。しかし，どちらかの引数が，対応する記述子の中の属性をもっていないときは，選択処理は，記述子を2つだけもつ次の総称メンバーについて調べる。たとえば，次のようなステートメントについて考えてみよう。

```
DECLARE CALC GENERIC
        (FXDCAL WHEN (FIXED, FIXED),
        FLOCAL WHEN (FLOAT, FLOAT),
        MIXED WHEN (FLOAT, FIXED));
```

このステートメントは，FXDCAL，FLOCAL，および MIXED という3つのメンバーをもつ総称名としてCALCを定義している。CALCに対して総称参照を行なうと，その参照の中にある2つの引数の性質に応じて，これらの3つの関数プロシージャーのうちの1つが呼び出される。たとえば，次のようなステートメントについて考えてみよう。

```
Z = X + CALC(X,Y);
```

XとYがそれぞれ浮動小数点と固定小数点の場合は，MIXEDが呼び出される。

同様にして，プロシージャーへの入口点は，次元付けの方法を用いて選択することもできる。たとえば，

```
DCL D GENERIC(D1 WHEN((*)),
              D2 WHEN((*,*))),
    A(2),
    B(3,5);
CALL D(A);
CALL D(B);
```

最初の呼出しステートメントが実行されると，プロシージャー入口点D1が呼び出される。2番目の呼出しステートメントが実行されると，プロシージャー呼出し点D2が呼び出される。

記述子がすべて省略されているか，または星印になっている場合は，現在の数の記述子をもつ最初の入口名が選択される。

特定の総称関数参照に対して，引数の属性が一致する総称記述子リストが何もみつからない場合は，プログラムはエラーになる。

組込み関数

プログラマーの書いたプロシージャーに対する関数参照のほかに，前もって定義された，広範囲にわたる関数のセット（組込み関数と呼ばれる）の1つを，関数参照によって呼び出すことができる。

組込み関数は，PL/Iの一部として備わっているものである。この中には，一般によく用いられる算術関数ばかりでなく，言語のもつ機能（たとえば，ストリングや配列の取扱い機能など）に関連して必要であったり，役に立つような関数も含まれている。

組込み関数は，プログラマーが定義した関数が呼び出される場合と同様の方法で呼び出される。しかし，プログラマーが定義した関数は1個の要素値しか返すことができないのに対して，組込み関数の多くは，値の配列を返すことができる。

注：　ある種の関数は，プロシージャーの呼出しとしてではなく，イン・ラインのコードとして実際にコンパイルされる。

引数リストをもつ組込み関数（SUBSTR（X, Y, Z）やINDEX（A, 'B'）など）を使用するときは，組込み関数として識別名を設定するのに必要な識別をそれ以上しなくても，組込み関数として認められる。しかし，括弧に入れた引数リストをもたない組込み関数や擬似変数の場合（ONCHAR, ONSOURCE, TIME など）は，その識別名が組込み関数として使用されているブロックの中で，BUILTINという属性をつけて明示的に宣言するか，あるいはナル引数リストをつけて（TIME()というような形で）指定しなければならない。

組込み関数名は，プログラマー定義の名前として使用することができる。したがって，組込み関数参照が，別のブロックに含まれているブロックの中で用いられており，しかもその別のブロックでは同じ識別名が何か他の目的に宣言されているような場合は，この参照は不明確なものになる。このような不明確さを避けるために，その識別名に関する別の宣言を外側のブロックから受けついでいるブロックの中で，組込み関数名に対して BUILTIN 属性を宣言することができる。次の例について考えてみよう。

```
A: PROCEDURE;
     ⋮
   B: BEGIN;
      DECLARE SQRT FLOAT BINARY;
        ⋮
      C: BEGIN;
         DECLARE SQRT BUILTIN;
           ⋮
         END C;
        ⋮
      END B;
     ⋮
   END A;
```

外部プロシージャーAの中で，SQRTは，BUILTIN属性をもつものと文脈的に宣言されているとしよう。したがって，SQRTを参照すればつねに，その名前の組込み関数が参照されることになる。しかし，Bの中では，SQRT は浮動小数点2進数の変数と宣言されており，他の方法で使用することはできない。最後に，Cの中では，SQRT は BUILTIN 属性をつけて宣言されているので，SQRT への参照は，Bの中で宣言された浮動小数点2進数の変数ではなく，組込み関数を参照したものと認められる。

組込み関数と同じ識別名をもつ変数は，算術変数として暗示的に宣言することができることに注意していただきたい。たとえば，割当て記号の左側（割当てステートメント，DOステートメント，繰返し指定の中の）や，GETステートメントのデータ・リストの中（ただし，引数リストの括弧の中または引数リストの直前にあるのでないことが条件）に，この識別名がある場合などが，これにあたる。（このことは，引数を必要としない擬似変数であるONCHARおよびONSOURCEなどの名前にも適用される。）たとえば，開始ブロックBの中に，DECLAREステートメントのかわりに，

SQRT=1 というステートメントがあったとすれば，SQRT は，浮動小数点10進数変数として暗示的に宣言されたことになる。

プログラマーは，組込み関数名を，プログラマー定義の関数の入口名としても使用することができるし，しかも，同じプログラムの中でこの関数名を，組込み関数とプログラマー定義の関数の両方に使用することさえもできる。プログラマー定義の関数が内部プロシージャーであるときはBUILTIN属性を使用し，プログラマー定義の関数が外部プロシージャーであるときはBUILTINとENTRY 属性を使用すれば，上記のことが可能になる。

次の例は，内部関数プロシージャーと結びつけてBUILTIN属性を使用する例である。

```
    A: PROCEDURE;
    SQRT: PROC(PARAM) RETURNS(FIXED(6,2));
          DECLARE PARAM FIXED (12);
            :
          END SQRT;
            :
      X = SQRT(Y);
            :
      B:  BEGIN;
          DECLARE SQRT BUILTIN;
            :
          Z = SQRT (P);
            :
          END B;
            :
       END A;
```

2番目のPROCEDUREステートメントのラベルとしてSQRTを使用しているので，この識別名は入口名として明示宣言されたことになる。したがって，Aの中の割当てステートメントでこの関数を参照すると，プログラマー作成のSQRT 関数を参照することになる。開始ブロックBの中では，識別名SQRTはBUILTIN属性をつけて宣言されている。したがって，Bの中にある割当てステートメントの関数参照は，組込み関数のSQRT を参照することになる。

この組込み関数の名前を用いるプログラマー作成の内部関数の場合，外側のブロックの中でこの識別名を参照すると，プログラマー作成の関数を参照することになる。上の例では，返される値の属性は，SQRT に対するプロシージャー・ステートメントのRETURNSオプションの中で指定されている。この関数参照は内部のものなので，これらの属性は呼出し側プロシージャーに知られている。

入口名として組込み関数の名前を使用する，プログラマー作成の外部関数プロシージャーの場合は，その関数プロシージャー名への参照を含んでいるプロシージャーには，その名前の入口宣言も含めなければならない。このようにしないと，この識別名への参照は，組込み関数に対するものになる。上の例の中で，開始ブロックBがAの中に含まれているのでなければ，BUILTIN属性を指定する必要はなく，この識別名が BUILTIN 以外の属性をつけて宣言されているのでない限り，組込み関数が参照されることになる。
プロシージャーSQRT が外部プロシージャーの場合は，SQRT を入口名として明示的に宣言するため，およびプログラマー作成の関数プロシージャーに渡される値およびそこから返される値の属性を指定するために，プロシージャーAでは次のようなステートメントが必要である。

```
    DCL SQRT ENTRY(FIXED(12))
             RETURNS(FIXED(6,2));
```

FORTRANライブラリー関数

FORTRANコンパイラーには，PL/Iの組込み関数とよく似たライブラリー関数があり，これらの関数は，PL/Iの言語間連絡機能(第17章でのべる)を用いて，PL/Iプログラムから呼び出すことができる。

組込みサブルーチン

PL/Iのプログラマーは，組込みサブルーチンを用いて，ある種のオペレーティング・システムの機能を，自分のプログラムの中で役に立てることができる。これらは，各自のコンパイラーで定義された入口名をもっており，CALL ステートメントによって呼び出される。このようなオペレーティング・システムの機能と，対応する入口名は次の通りである。

チェックポイント／再始動：	PLICKPT
分類／組合せ：	PLISRTA PLISRTB PLISRTC PLISRTD

このほかにも，主記憶域のダンプ(編集したもの)をとるためのサブルーチンであるPLIDUMP，プログラムの戻りコードをユーザーがセットするためのサブルーチンであるPLIRETCがある。これらの入口名は，組込み名として知られており，BUILTIN属性をもつことを，明示的または文脈的に宣言することができる。

これらは予約語ではない。

これらのサブルーチンの用法については，このコンパイラー用の「プログラマーの手引書」をみていただきたい。

引数とパラメーターの関係

関数またはサブルーチンが呼び出されると，呼出しをしたステートメントまたは式の引数と，呼び出された入口点のパラメーターとの間で関連づけが行なわれる。この関連づけは，見かけ引数が作られるかどうかによって異なる。

見かけ引数

前に引数とパラメーターについて説明した際に，サブルーチンまたは関数に渡されるのは，引数の値ではなく，引数の名前であるということを指摘した。しかし，これはつねに可能なわけではない。たとえば，定数や演算式は名前をもっていない。したがって，コンパイラーは，このような値に対して記憶域を準備し，対応するパラメーターの名前とそれらをそれぞれ結びつける。これらの記憶位置は，見かけ引数と呼ばれる。パラメーターに対する変更は，見かけ引数の値に対してだけ反映され，それが作られるもとになった引数の値には影響しないので，プログラマーはこのような存在について気使う必要はない。

もとになる引数が，次のいずれかであるときは，見かけ引数がつねに作成される。

1. 定数

2. 演算式を含む式

3. 括弧に入っている式

4. パラメーターに対して宣言されたデータ属性とは違うデータ属性をもっている変数。これは，パラメーターのストリングの長さまたはエリア・サイズを定義するために星印が使用されているときは適用されない。このような場合は，コンパイラーは，引数とパラメーターが合致しているものとみなす。（ PICTURE 属性をもつ引数とパラメーターの場合は，ピクチャーの指定が完全に一致していない限り，繰返し因数が適用されたあとで，見かけ引数が作られる。ただし，唯一の例外として，スケール因数の中に＋符号をもつ引数またはパラメーターは，＋符号をもたないパラメーターまたは引数と同じとみなされる。）

5. 引数リストをもつ関数参照

6. 被制御または調整可能な範囲をもつストリングまたはエリアのうち，一定の範囲をもち，被制御でないパラメーターと結びつけられているもの。

7. i SUB 定義の配列

内部プロシージャーに渡される引数に対して作られる見かけ引数の属性は，次のようにして推論される。

1. 内部プロシージャーの中で関連するパラメーターに対して宣言されている属性から。

2. 配列の限界，ストリングの長さ，エリアのサイズが，パラメーターの宣言の中で星印によって指定されている場合は，引数自体の限界，長さ，サイズから。

上記以外の場合はすべて，引数の参照は直接渡される（実際に，引数の記憶域アドレスが渡される）。パラメーターは，渡された引数と全く同じになるので，パラメーターの値の変更がもとになる引数の値に影響するのは，見かけ引数が渡されないときだけである。

ENTRY属性

ENTRY 属性は，外部プロシージャーまたは入口変数の名前を識別するために用いられる。

ENTRY 属性の一般形式については，この章の前の方の「ENTRY属性の用法」のところで説明した。この形式では，引数を必要としない関数入口名を識別するのに用いられるとき，または引数とパラメーターが一致するときは，パラメーター記述子リストをつけずに ENTRY というキーワードを指定することが許されることに注意していただきたい。引数とパラメーターが一致しないときは，外部プロシージャーの入口名を識別する ENTRY 属性をつけて，パラメーター記述子リストを指定しなければならない。入口宣言の中でVARIABLE属性を使用すると，その識別名は入口変数として設定される。入口変数は，入口定数が入口変数に割り当てられたあとの入口定数を表わしている。引数が渡される入口点を呼び出す

ために，関数参照またはCALL ステートメントの中で入口変数が使用されている場合は，入口点のパラメーターの属性を指定するパラメーター記述子リストをつけて入口変数を宣言しなければならない。このようにしないと，誤った引数が渡されることもある。

パラメーター記述子リスト

ENTRY 属性指定の中のパラメーター記述子リストの中にある属性のセット，すなわち記述子はそれぞれ，呼び出されるサブルーチンまたは関数のパラメーターの1つ1つに対応しており，与えられている場合には，そのパラメーターの属性を指定する。個々のパラメーターに対するいくつかの属性を，ブランクで区切って，各々のパラメーターに対するパラメーター記述子を構成する。パラメーター記述子リストの中のパラメーター記述子は，カンマによって区切る。一般に，引数の属性が，それに対応するパラメーターの属性（パラメーター記述子リストで指定されているもの）と一致しない場合は，変換が可能であれば，その引数に対して見かけ引数が作られる。見かけ引数には，もとになる引数の値を対応するパラメーターの属性と合うように変換したものが入っている。したがって，サブルーチンまたは関数が呼び出されたときに，そこに渡されるのは，見かけ引数である。

ENTRY 属性といっしょに記述子リストを与える場合はサブルーチンまたは関数の各パラメーターもその中に入れて考えなければならない。引数とパラメーターの属性が一致しているときは，記述子を星印で表示しても，省略してもよいが，記述子を区切るカンマは省略してはならない。たとえば，次のステートメントは，

```
DECLARE SUBR ENTRY (FIXED,,FLOAT);
```

SUBR は，3つのパラメーターをもつ入口点であることを指定している。最初と3番目の属性はそれぞれ，FIXEDとFLOAT という属性をもっているが，2番目の属性は渡された引数の属性と同じであるとみなされる。2番目のパラメーターの属性は記述されていないので，想定は何も行なわれない。

前に述べたように，ENTRY 属性は，パラメーター記述子リストをつけずに指定することもできる。この属性は，結びつけられた識別名が入口名であることを表示するために，このようにして用いられる。ある識別名が，他の方法では入口名として認めさせることができない場合，つまりPROCEDUREまたは ENTRY ステートメントのラベルとしてその識別名を記入することによって，その識別名を入口名として明示宣言しないような場合に，このような表示が必要である。

したがって，入口名がこのような形で出てきていないようなブロックの中で参照が行なわれる場合は，識別名には ENTRY 属性を明示的に与えなければならない。たとえば，次のものが指定されていたものとし，

```
A:   PROCEDURE;
        ⋮
     PUT LIST (RANDOM);
        ⋮
     END A;
```

さらにまた，Aは外部プロシージャーで，RANDOM は引数を必要とせず，乱数を返すような外部関数であるとしよう。このプロシージャーは上に示した通りであるから，RANDOM は，Aの中では入口名として認められず，したがって当然，PUTステートメントの結果は定義されていないことになる。Aの中でRANDOM が入口名として認められるようにするためには，ENTRY 属性をもつようRANDOM を宣言しなければならない。たとえば，

```
A:   PROCEDURE;
     DECLARE RANDOM ENTRY;
        ⋮
     PUT LIST (RANDOM);
        ⋮
     END A;
```

これで，RANDOM は入口名として認められ，PUTステートメントの中にRANDOM があれば，関数参照以外の何ものとしても解釈することはできない。したがって，PUTステートメントがあると，RANDOM から返される乱数が出力転送される。

<u>注</u>：DECLAREステートメントの中で入口名属性 RETURNS または OPTIONS の1つをもつと宣言されている識別名に対しては，ENTRY 属性が暗示される－－したがって，明示的に記述する必要がない。

引数としての入口式

関数またはサブルーチン参照の引数として入口式が指定されているときは，次のいずれかが適用される。

1. 入口式の引数（Mと呼ぶことにする）がそれ自身の引数リストをもって指定されている場合は，これは関数参照として認められる。Mが呼び出され，Mから返される値が実際に，それを含んでいる引数リストの中で，Mおよびその引数リストと置き換わる。たとえば，

例：

```
CALL A (M(B));
```

これは，関数プロシージャーMから返される値を渡すものである。

入口式引数がナル引数リストをつけて記入されている場合は，これは，引数をもたない関数参照として受けとられる。たとえば，

```
CALL A(B());
```

これは，関数プロシージャーBから返される値を，プロシージャーAへの引数として渡すものである。

2. 入口式引数が引数をもっておらず，しかも括弧の中に入っている場合は，見かけ入口変数が作られる。たとえば，

```
CALL A((B));
```

これは，入口名Bの値を，プロシージャーAへの引数として渡すものである。

3. 組込み関数名または入口式が，組込み関数に対する引数として，引数リストをつけずに使用されている場合は，その組込み関数がENTRYタイプの引数を受け入れるならば，その引数によって指定された関数は呼び出されない。その組込み関数が入口引数を受け入れないときは，その引数は，その関数の値を参照するものとみなされる。たとえば，次の場合は，

```
DCL DATE BUILTIN, Z CHAR(2);
Z = SUBSTR (DATE, 5, 2);
```

DATE組込み関数から返される値から，日付フィールドがとり出される。

4. ユーザー定義の関数に対する入口式引数が，引数リストをもっておらず，しかも，演算式または括弧の中のどちらにあるのでもないときは，入口式自体が，呼び出される関数またはサブルーチンに渡される。このような場合，この入口式は，引数を必要としないような関数の名前であっても，関数参照とは受けとられない。たとえば，

```
CALL A(B);
```

これは，プロシージャーAに対する引数として入口式Bを渡すものである。Aの中の対応するパラメーターがENTRY属性をつけて宣言されている場合は，省略時解釈によりVARIABLE属性が与えられる。Bが入口変数の場合は，パラメーターの属性と一致する属性をもつ引数の場合と同じようにして，Bがパラメーターに渡される。Bが入口定数の場合は，定数の引数と同じようにして，見かけが作られ，渡される。

しかし，この規則には例外がある。識別名が入口名として知られており，しかも引数として出てきている場合，およびその引数に対するパラメーター記述子が，ENTRY以外の属性を指定している場合は，その入口名が呼び出され，そこから返された値が渡される。返された値がパラメーター記述子の中で指定された属性とは異なる属性をもっている場合は，変換が行なわれる。たとえば，

```
A: PROCEDURE;
   DECLARE B ENTRY,
           C ENTRY(FLOAT);
     ⋮
   X = C(B);
     ⋮
   END A;
```

この場合は，Bが呼び出され，そこから返される値がCに渡される。

次の例について考えてみよう。

```
CALLP: PROCEDURE;
       DECLARE RREAD ENTRY,
               SUBR ENTRY (ENTRY, FLOAT,
                  FIXED BINARY, LABEL);
         :
       GET LIST (R,S);
         :
       CALL SUBR (RREAD, SQRT(R), S,
          LAB1);
         :
 LAB1: CALL ERRT(S);
         :
       END CALLP;
 SUBR: PROCEDURE(NAME, X, J, TRANPT);
       DECLARE NAME ENTRY, TRANPT LABEL;
         :
       IF X > J THEN CALL NAME(J);
              ELSE GO TO TRANPT;
         :
       END SUBR;
```

この例の中で，CALLP，SUBR，およびRREADは，外部であるとしよう。CALLPの中で，RREADとSUBRの両方がENTRY属性をもつものと明示宣言されている。SUBRのパラメーターの性質に関する情報を与えるために，SUBRに対して明示宣言が用いられている。CALL SUBRステートメントの中では，4つの引数が指定されている。これらの引数は，次のように解釈される。

1. 最初の引数，RREADは，入口名として認められる（ENTRY属性の宣言があるので），この引数は，CALLPの中でのSUBRに対するENTRY属性の宣言の中で指定されている最初のパラメーター記述子とは矛盾していない。したがって，RREADは関数参照ではなく入口名として認められるので，呼出しの位置で入口名が渡される。NAMEは入口パラメーターなので，省略時解釈によってVARIABLE属性が与えられる。RREADは定数なので，見かけの入口引数が作られ，これがNAMEに渡される。

2. 2番目の引数，SQRT(R)は，引数リストの後に入口名があるので，組込み関数参照として認められる。SQRTが呼び出され，SQRTから返される値が見かけ引数に割り当てられ，これがサブルーチンSUBRに渡される。見かけ引数の属性は，パラメーター属性リストの宣言の中で指定された通り，2番目のパラメーターの属性と一致している。SUBRが呼び出されると，見かけ引数がSUBRに渡される。

3. 3番目の引数，Sは，単なる10進浮動小数点要素変数である。しかし，その属性は3番目のパラメーターの属性とは一致していないので，3番目のパラメーターの属性に合わせてSの値を変換したものを含む見かけ引数が作られる。SUBRが呼び出されると，見かけ引数が渡される。

4. 4番目の引数，LAB1は，ステートメント・ラベル定数である。その属性は，4番目のパラメーターの属性と一致している。しかし，これは定数なので，見かけ引数が作られる。SUBRが呼び出されると，見かけ引数が渡される。

SUBRの中で，4つのパラメーターは，PROCEDUREステートメントによって明示的に宣言されている。これらのパラメーターに対してそれ以上明示宣言が与えられていなければ，省略時解釈の算術属性がそれぞれに対して与えられる。したがって，NAMEは入口名を表わしていなければならないので，NAMEはENTRY属性をつけて明示的に宣言されており，TRANPTはステートメント・ラベルを表わしていなければならないので，これはLABEL属性をつけて明示的に宣言されている。XとJは算術タイプなので，省略時解釈を適用することができる。

CALLステートメントの中にNAMEがあっても，組込みプロシージャーとしてNAMEを文脈宣言したことにはならないことに注意していただきたい。明示宣言が適用されなければ，このような文脈宣言が行なわれる。しかし，SUBRのPROCEDUREステートメントの中にNAMEがあるということは，パラメーターとしてNAMEを明示宣言したことになる。補足をするDECLAREステートメントの中でパラメーターの属性が明示的に宣言されていないときは，算術の省略時解釈が適用される。したがって，NAMEは，ENTRY属性をもつものとして明示的に宣言しなければならない。このようにしないと，NAMEは，2進固定小数点変数とみなされることになり，CALLステートメントの中で使用すると，エラーになる。

パラメーターの割振り

パラメーターは，呼び出されたプロシージャーの中にはそれと結びつけられた記憶域をもたないので，これに対して，記憶域クラスの属性STATIC，AUTOMATIC，またはBASEDのいずれも宣言することができない。しかし，CONTROLLED属性をもつと宣言することはできる。したがって，記憶域の割振りに関しては，2つのクラスのパラメーター ― 記憶域クラスをもたないもの，すなわち単純パラメーター，とCONTROLLED属性をもつもの，すな

わち被制御パラメーター――があることになる。

単純パラメーターは，どのような記憶域クラスをもつ引数とでも結びつけることができる。しかし，引数について２つ以上の世代が存在している場合は，パラメーターは，呼出しの時点に存在しているその世代とだけ結びつけられる。

被制御パラメーターはつねに，対応する被制御引数をもっていなければならない。このような引数は，添字をつけることはできないし，構造の要素であってはならないし，また，見かけ引数が作られるようなものであってはならない。呼出しの時点で引数の２つ以上の世代が存在する場合は，パラメーターは，これらの世代の積重ね全体に対応する。このように，呼出しの時点では，被制御パラメーターは対応する引数の現在の世代を表わしている。被制御パラメーターは，呼び出されたプロシージャーの中で割り振ったり，解放したりすることができるので，結びつけられた引数の積重ねの割振りを操作することができる。単純パラメーターは，ALLOCATEまたはFREE ステートメントの中で指定することはできない。

パラメーター記述子が与えられていない場合は，積重ね全体が渡される。この場合のパラメーターは，単純でも被制御でもよく，最新の世代または積重ね全体のどちらに対応して結びつけられていてもよい。

パラメーターの属性

DEFINEDまたはBASED属性をつけて，パラメーターの宣言をすることはできない。パラメーターは，オーバーレー定義をするための基底識別名として用いることもできるし，また，CONNECTED属性をもっているときは， レコード方式の転送のために使用することもできる。パラメーターはつねにINTERNAL属性をもっており，レベル－１の識別名でなければならない。

パラメーターの限界と範囲

引数が，配列，ストリング，またはエリアのいずれかである場合は，対応するパラメーターに関して，配列の限界，またはストリングやエリアの範囲を宣言しなければならない。配列パラメーターの次元数と限界，またはストリングやエリア・パラメーターの範囲は，対応する引数の現在の世代のそれと同じでなければならない。普通，パラメーターの限界または範囲の実際の数を指定しさえすれば，上記のことは守られるはずである。サブルーチンまたは関数を作成する段階で，限界や範囲がわからなければ，単純パラメーターのときは星印によって，被制御パラメーターのときは星印または式によって指定すればよい。

単純パラメーターの限界，長さ，およびサイズ

単純パラメーターの実際の限界または範囲が，呼出しごとに違っている可能性がある場合は，DECLAREステートメントの中で星印を使ってこれらを指定することができる。星印が使用されているときは，対応するPL／I 引数の現在の世代から限界または範囲がとられる。現在の世代が存在していないときは，その変数に対する参照はエラーになる。

被制御パラメーターの限界と範囲

被制御パラメーターの限界または範囲は，DECLAREステートメントの中で星印または要素式を用いて表わすことができる。

星印表記： 星印表記が用いられているときは，被制御パラメーターの限界または範囲は，対応する引数の現在の世代からとられる。その後，被制御パラメーターの割振りは，ALLOCATEステートメントの中で異なる限界または範囲が指定されて，それらが置き換えられない限り，上記のものと同じ限界または範囲を使用する。引数の現在の世代が存在していないときは，星印は，そのパラメーターが次元をもつということしか決められないので，パラメーターに対する他の参照が行なわれないうちに，呼び出されるプロシージャーの中でALLOCATEステートメントを使って，被制御パラメーターに対する限界と範囲を指定しなければならない。

式表記： 被制御パラメーターの限界または範囲，またはサイズは，要素式によって指定することもできる。これらの式は，割振りの時点で評価される。パラメーターが割り振られるたびに，新しい割振りに対する現在の限界または範囲を与えるために，式が再評価される。しかし，DECLAREステートメントの中のこのような式は，ALLOCATEステートメント自体の中で限界または長さを指定することによって置き換えることができる。

パラメーターが割り振られるたびに式が評価される。ただし，ALLOCATEステートメント自体の中で限界または範囲が指定されて，これらが置き換えられる場合は例外である。たとえば，．

```
MAIN:  PROCEDURE OPTIONS(MAIN);
       DECLARE (A(20),B(30),C(100),
                D(100))CONTROLLED,
                NAME CHARACTER (20),
                I FIXED(3,0);
          :
       ALLOCATE A,B;
       CALL SUB1(A,B);
          :
       FREE A,B;
       GET LIST (NAME, I);
       CALL SUB2 (C,D,NAME,I);
          :
       FREE C,D;
          :
       END MAIN;
SUB1:  PROCEDURE (U,V);
       DECLARE (U(*), V(*)) CONTROLLED;
          :
       ALLOCATE U(30), V(40);
          :
       RETURN;
       END SUB1;
SUB2:  PROCEDURE (X,Y,NAMEA, N);
       DECLARE (X(N), Y(N))CONTROLLED,
                NAMEA CHARACTER (*),
                N FIXED(3,0);
          :
       ALLOCATE X, Y;
          :
       RETURN;
       END SUB2;
```

プロシージャーMAINの中で，配列A，B，CおよびDは，CONTROLLED記憶域クラス属性をつけて宣言されており，NAMEとIは，省略時解釈によりAUTOMATICになる。

SUB1が呼び出されると，AとBは宣言された通りに割り振られてから，渡される。SUB1は，星印表記によってそのパラメーターを宣言している。しかし，ALLOCATEステートメントが，配列の限界を指定しているので，割り振られた配列（実際には，AとBの2番目の世代）は，最初の世代とは違った限界をもっている。ALLOCATEステートメントの中で限界が指定されなかった場合は，新しい世代の限界は，最初の世代の限界と同じになる。

MAINに制御が戻ったあと，最初のFREEステートメントはAとBの2番目の世代（パラメーターとしてSUB1の中で割り振られたもの）を解放し，2番目のFREEステートメントは最初の世代（MAINの中で割り振られたもの）を解放する。

SUB2が呼び出されると，CとDがXとYに渡され，NAMEがNAMEAに渡され，IがNに渡される。SUB2の中で，I（Nに渡される）の値によって決まる限界が，XとYに対して，宣言されている。XとYが割り振られるとき，この値によって，割り振られた配列の限界が決まる。

NAME（NAMEAに対応する）は被制御ではあるが，NAMEAの長さに対して星印が表記されているということは，その長さを，引数NAMEから取り出すことを表わしている。

引数およびパラメーターのタイプ

一般に，引数とそれに対応するパラメーターは，どのようなデータ編成やタイプのものであってもよい。しかし，パラメーターと引数の関係がすべてはっきりとしたものであるとは限らない。さらに定義づけや明確化が必要なものもある。このような場合について以下に述べる。

パラメーターが要素のとき，つまり，構造でも配列でもない変数のときは，引数は要素式でなければならない。引数が添字付の変数のときは，サブルーチンまたは関数が呼び出される前に添字が評価され，指定された要素の名前が渡される。外部プロシージャーに渡される引数が定数のときは，入口宣言の中に対応するパラメーター記述子がない限り，対応するパラメーターの属性は，その定数によって示された属性と一致していなければならない。

パラメーターが配列のときは，引数は配列式でも要素式でもよい。引数が要素式の場合は，対応するパラメーター記述子または宣言によって，配列パラメーターの限界を指定しなければならない。10進整定数として，限界を指定しなければならない。このようにすると，配列パラメーターの限界と同じ限界をもつ見かけ配列引数が作られる。このあと，要素式の値が，見かけ配列引数の各要素の値に割り当てられる。

パラメーターが固定長ストリングで，しかも見かけ引数が作られない場合は，引数も固定長ストリングでなければならない。同様にパラメーターが長さの変わるストリングで，見かけ引数が作られない場合は，引数も長さの変わるストリングでなければならない。

長さの変わる要素ストリング引数が，長さが定義されてはいない（つまり，星印で指定された）が，長さの変わるものではない要素ストリング・パラメーターに渡されるときはつねに，引数の現在の長さが，呼び出されるプロシージャーに渡される。引数が，長さの変わるストリング配列であり，しかもそれが，長さの変わるものではないが，長さが定義されていないパラメーターに渡される場合は，1つの長さだけ，つまり最大の長さだけが渡される。

パラメーターが構造の場合は，引数は構造式または要素式のどちらかでなければならない。引数が要素式のときは，外部入口点の対応するパラメーター記述子は，構造パラメーターの構造記述子を指定するものでなければならない（レベル番号だけを使用すればよい──詳細については，I節「属性」のところでENTRY属性の説明をみていただきたい）。この結果，構造パラメーターの記述と一致する記述をもつ見かけ構造引数が作られることになる。そして，要素式の値は，見かけ構造引数の各要素の値になる。引数の構造とパラメーターの構造は，相対的に同じでなければならないが，レベル番号は同じでなくてもよい。要素の値は，構造のすべての要素名の属性に合わせて，変換可能なものでなければならない。

パラメーターが構造の配列である場合は，引数は，要素式，配列式，構造式，構造の配列式，のいずれであってもよい。

パラメーターがラベルの場合は，引数はラベル変数かラベル定数のどちらかでなければならない。引数がラベル定数の場合は，見かけ引数が作られる。

パラメーターが入口の場合は，引数は入口式または総称名でなければならない。引数が総称名の場合は，入口を渡す前に総称選択を行なうことができるように，パラメーター記述子（呼び出されるプロシージャーが内部のときはパラメーター宣言）によって，パラメーター記述を与えなければならない。引数として渡される入口変数は，引数とパラメーターが境界合せの点でだけ違っているときは見かけ引数を作らずにすむように，境界合せされているものとみなされる。算術組込み関数の名前は引数として渡すことができるが，その他の組込み関数名は渡すことができない点に注意していただきたい。

パラメーターがファイルの場合は，引数はファイル式またはファイル定数でなければならない。たとえば，

```
E:      PROCEDURE;
        DECLARE F1 FILE;
        CALL E1(F1);
          :
E1:     PROCEDURE(F2);
        DECLARE F2 FILE;
        CALL E2(F2);
          :
E2:     PROCEDURE (F3);
        DECLARE F3 FILE;
          :
        END E;
```

ファイル・パラメーターF1，F2，およびF3はすべて，同じファイルを参照している。これらのファイル・パラメーターに対する入出力のオン・ユニットについては，第12章「例外条件の取扱いおよびプログラム・チェックアウト」のところで説明する。

パラメーターが，ポインターまたはオフセットのどちらかのタイプのロケーターである場合は，その引数は，どちらかのタイプのロケーター式でなければならない。タイプが違っているときは，見かけ引数が作られる。オフセット・パラメーターのパラメーター記述子は，結びつけられるエリアを指定してはならない。

パラメーターがエリアの場合は，引数は要素式でなければならない。両方のサイズが違っているときは，見かけ引数が作られる。

第10章： 入力と出力

PL/Iには，コンピューターの内部記憶装置と外部記憶装置の間でデータを転送するための入出力ステートメントがある。プログラムの外部にあるデータの集まりは，データ・セットと呼ばれる。データ・セットからプログラムへデータを転送することは，入力と呼ばれ，プログラムからデータ・セットへデータを転送することは，出力と呼ばれる。

PL/Iの入出力ステートメントは，データ・セットの論理的な構成に関係するものであり，プログラムが実行されるときに使用される入出力装置についての特別な知識を必要とせずに，プログラムを書くことができる。原始プログラムが，データ・セットの中の物理的な構成ではなく，データの論理的な面を主として取り扱うことができるように，PL/Iは，ファイルと呼ばれるデータ・セットの記号表現を使用している。ファイルは，プログラムの実行中に，時点が違えば別のデータ・セットと結びつけることができる。

PL/Iのプログラムでは，2つのタイプのデータ転送を使用することができる。

ストリーム方式の転送では，データ・セット内におけるデータの構成は，プログラムの中では無視され，データは，実際に文字形式をした個々のデータ項目の連続した流れ（ストリーム）であるとして扱われる。入力のときは，データは文字形式から内部形式に変換され，出力のときは，内部形式から文字形式に変換される。

レコード方式の転送では，データ・セットは，不連続のレコードの集まりとみなされる。レコードの転送中には，データ変換は行なわれない。入力のときは，データは，データ・セットの中に記録されている通りの形で転送され，出力のときは，内部的に記録されている通りの形で転送される。（ただし，ASCIIデータ・セットの場合は，この通りではない。この章の後の方の「情報交換コード」の項をみていただきたい。）

別の時点であれば，同じデータ・セットを，ストリーム転送とレコード転送のどちらででも処理することができる。しかし，そのような場合は，データ・セット内のすべての項目が文字形式になっていなければならない。

ストリーム方式転送は，単純な仕事，とくに，入力が穿孔カードで，出力が限定されている，というような仕事の場合に，非常に向いている。とくに，入力が穿孔カードで，出力が印刷の場合には，プログラマーのコーディングの手間は最少ですむ。しかし，レコード方式の転送にくらべて，ストリーム方式の転送は，実行時間の点で効率が悪い。これは，この方式にはデータ変換が含まれることと，すべてのデータが文字形式であるために，外部記憶装置上に余分なスペースが必要であることが原因である。

レコード方式転送は，ストリーム方式転送にくらべて，データを処理する方式と，処理可能なデータ・セットのタイプの両面に関して，より融通性に富んでいる。データは，主記憶装置にある通りの形でデータ・セットの中に記録されるので，どのようなデータ形式も受け入れられる。したがって，変換の問題は起らないが，プログラマーは，データの構成に充分注意を払わなければならない。

この章では，ストリーム方式とレコード方式の転送に共通なPL/Iの入出力の様々な性質（ファイルおよびその属性，ファイルとデータ・セットの関係なども含めて）について解説する。この後の2つの章では，PL/Iのプログラムの中で使用できる入出力ステートメントと，PL/Iで認められる各種のデータ・セット編成について解説する。

データ・セット

データ・セットは，様々な補助記憶媒体，たとえば穿孔カード，磁気テープのリール，磁気ディスク，および磁気ドラムなどに貯えられる。その種類は各様であるが，これらの媒体の性質は，共通な面も多くもっているので，データの蒐集，記憶，転送に関して標準的な方法を設けることも可能である。便宜上，補助記憶装置の1単位（たとえば，磁気テープのリールや，ディスク・パックなど）を指すのに，その特別な物理的な構成には関わりなく，ボリュームという一般的な用語が用いられる。

データ・セット内の各データ項目は，ブロックと呼ばれる物理的なグループに分けて編成されている。このようなブロックがあるために，データ・セットは1つの単位としてではなく，部分に分けて転送したり，処理したりすることができる。さらに，処理に都合のよいように，各々のブロックは，レコードと呼ばれる1つまたは2つ以上の論理的な小部分（その各々には，1つまたは2つ以上のデータ項目が含まれる）から構成してもよい。（ブロックは，ボリュームとの間で物理的に転送されるデータの単位であるため物理レコードと呼ばれ，その論理的な小部分は論理レコードと呼ばれることもある。）

ブロックに2つ以上のレコードが含まれているときは，そのレコードはブロック化されているという。レコードをブロック化すると，外部記憶装置は，密度が高くなり，効率よく使用することができるようになる。短いレコードを大量に処理する場合は，ブロック化レコードを使用すると，物理的な入出力操作の回数が減るので，プログラムのスループットも上がる。

データ処理業務で扱われるのは，殆んどの場合，ブロックではなく，論理レコードである。そのため，PL/Iの入出力ステートメントは，一般に，論理レコードを参照する。したがって，プログラマーは，処理されるデータにのみ専念すればよく，外部記憶装置上での物理的な構成に直接わずらわされることがない。

情報交換コード

システム/360およびシステム/370では，データの表現に用いられる標準コードは，主記憶装置上でも，補助記憶装置上でも，EBCDIC（拡張2進化10進コード）である。通常，このコンパイラーでコンパイルされたPL/Iプログラムは，EBCDIC を使ってすべての文字データを記録する。しかし，オペレーティング・システムは，補助記憶装置内のデータを表現するために，ASCII（情報交換用米国標準コード）と呼ばれるコードを使用することも許しており，PL/Iを使ってこのようなデータ・セットを読んだり，作成したりすることができる。このサポートは，磁気テープ上にあるデータ・セットに限られる。

2つのコードの間の変換は，オペレーティング・システムによって行なわれる。ENVIRONMENT属性で指定するオプションは別として，標準のEBCDIC データ・セットに対して使用したものと同じPL/Iプログラムを使用して，ASCIIデータ・セットを処理することができる。出力の場合は，データが緩衝域から外部記憶装置に書き出される直前に，EBCDIC から ASCII への変換が行なわれる。入力の場合は，緩衝域にデータが満たされるとすぐに，ASCII からEBCDIC への変換が行なわれる。

PL/Iの場合，ASCII データ・セット上に書くことができるのは，CHARACTERデータだけである。ASCIIコードの各文字は，7ビットのパターンで表わされるので，128個のパターンがある。EBCDIC では，各文字は8ビットのパターンで表わされるので，256のパターンがある。ASCIIデータ・セットには，該当する ASCII コードをもたないEBCDIC 文字を表すのに用いられる代用文字（SUB制御文字）が含まれている。このデータが読まれると，その文字はEBCDIC SUB文字（ビット・パターン00111111）に変換される。

ファイル

原始プログラムが，データ・セット内における物理的な構成ではなく，データの論理的な面を主として扱うことができるようにするために，PL/Iは，データ・セットに関して，ファイルと呼ばれる記号表現を使用している。それと結びつけられるデータ・セットを入出力ステートメントがどのように処理するかは，この記号表現によって決まる。しかし，データ・セットとは異なり，ファイルはその原始プログラムの中でだけ意味をもっているのであって，プログラムの外側で物理的な実体として存在しているわけではない。

PL/Iでは，ファイルを表わす識別名は，FILE 属性をつけて宣言されている必要がある。このような識別名は，ファイル定数でも，ファイル変数でもよい。ファイル変数は，ファイル定数を割り当てることのできるデータ項目である。割当ての後，このファイル変数を参照すると，割り当てられたファイル定数を参照するのと同じ意味をもつようになる。PL/Iプログラムによって処理されるデータ・セットはそれぞれ，ファイル定数識別名と結びつけなければならない。

ファイル定数： 各ファイルの個々の特徴は，ファイル属性と呼ばれるキーワードによって記述される。次にあげるリストは，各タイプのデータ転送に適用される属性を示すものである。

ストリーム方式の転送

FILE
STREAM
INPUT
OUTPUT
PRINT
ENVIRONMENT

レコード方式の転送

FILE
RECORD
INPUT
OUTPUT
UPDATE
SEQUENTIAL
DIRECT
BUFFERED
UNBUFFERED
BACKWARDS
KEYED
ENVIRONMENT

ファイル変数： ファイル変数は，属性FILE およびVARIABLE をもつ識別名であるが，FILE 以外のファイル属性をもつことはできない。ファイル変数を集めて配列または構造にすることができる。VARIABLE属性は，暗示することができる（たとえば，次元属性などによって）。

ファイル式： ファイル式は，ファイル定数，ファイル変数，またはFILE属性をもつ値を返す関数参照を参照するものであってよい。

各ファイル属性の詳細については，I節「属性」のところをみていただきたい。以下の説明は，各ファイル属性の簡単な説明とファイルに対するこれらの属性の宣言の仕方を示すものである。

ファイル属性

FILE属性は，それと結びつけられている識別名がファイルであることを示すものである。たとえば，次のステートメントの中で，識別名MASTERは，ファイル定数であると宣言されている。

```
DECLARE MASTER FILE;
```

次の例では，識別名 ACCOUNT はファイル変数であると宣言されており，ACCT1，ACCT2，……は，ファイル定数であると宣言されている。このあと，ファイル定数は，ファイル変数に割り当てることができる。

```
DECLARE ACCOUNT FILE VARIABLE,
        ACCT1 FILE,
        ACCT2 FILE,
          ⋮
```

次の例は，どのようにすれば，VARIABLE属性を暗示することができるかを示したものである。

```
DECLARE PAYREC(10) FILE;
```

PAYREC(I)－Iは1から10までの値をもつ－には，DECLARE ステートメントの中での，明示宣言によってFILE属性，次元属性(10)の暗示によってVARIABLE属性，が与えられる。

ファイル定数と結びつけられる属性は，択一的属性と付加的属性，の2つのカテゴリーにわけられる。択一的属性は，属性のグループの中から選ばれる属性である。グループ内の択一的属性の1つが明示宣言によっても暗示宣言によっても与えられておらず，しかも択一的属性の1つが必須である場合には，省略時解釈の属性が想定される。

付加的属性は，明示的に記述しなければならないか，あるいは明示的に記述された別の属性によって暗示される属性である。付加的属性 KEYEDは，DIRECT属性により暗示される。付加的属性 PRINT は，標準出力ファイル名SYSPRINTにより暗示される。付加的属性は，省略時解釈によって暗示することはできない。

注： すべての択一的属性と付加的属性は，FILE属性を暗示している。したがって，択一的属性または付加的属性がすでに1つでも明示的に指定されている場合は，FILE属性を指定する必要はない。

択一的属性

PL/Iには，択一的ファイル属性のグループが5つある。次にあげるのは，これらのグループのリストである。

グループ・タイプ	択一の属性	省略時解釈の属性
用途	STREAM|RECORD	STREAM
機能	INPUT|OUTPUT|UPDATE	INPUT
アクセス	SEQUENTIAL|DIRECT	SEQUENTIAL
緩衝技法	BUFFERED|UNBUFFERED	BUFFERED
有効範囲	EXTERNAL|INTERNAL	EXTERNAL

各グループ(有効範囲の属性はのぞく。これについては，I節「属性」のところで説明する)について，次に個々に説明する。

STREAM属性とRECORD属性

STREAM属性とRECORD属性は，そのファイルに対する入出力操作で使用されるデータ転送操作のタイプ(ストリーム方式かレコード方式か)を記述するものである。

STREAM属性は，文字形式でだけ記録されたデータ項目の連続する流れ(ストリーム)として，ファイルを取り扱うためのものである。

RECORD属性は，内部形式(どのようなタイプでもよい)で記録された1つまたは2つ以上のデータ項目からなる，一連のレコードの集まりとして，ファイルを取り扱うためのものである。

```
DECLARE MASTER FILE RECORD,
        DETAIL FILE STREAM;
```

INPUT，OUTPUT，およびUPDATE属性

これらの機能属性は，あるファイルに対して許されるデータ転送の方向を決めるものである。 INPUT属性は，読取りだけが行なわれるファイルに適用される。 OUTPUT属性は，これから作成される(データ・セットを拡張するという場合もある)ファイルに適用される。

UPDATE属性（RECORDファイルにだけ用いられる）は，入力と出力の両用に用いられるファイルを記述するものである。この属性を使用した場合は，既存のデータ・セットにレコードを挿入したり，すでにそのデータ・セットの中にあるレコードを変更したりすることができる。

SEQUENTIAL属性とDIRECT属性

これらのアクセス属性は，RECORD属性をもつファイルにだけ適用されるもので，ファイルの中のレコードをどのようにしてアクセスするかを記述するものである。

SEQUENTIAL属性は，物理的な順序かまたはキーの順序にしたがって，データ・セット内のレコードをアクセスすることを指定するものである。

DIRECT属性は，データ・セット内のレコードをどのような順序でアクセスしてもよいことを指定するものである。データ・セット内のレコードの位置は，文字ストリング"キー"によって決められるので，DIRECT属性はKEYED属性を暗示する。これと結びつけられるデータ・セットは，直接アクセス・ボリュームに入っていなければならない。

BUFFERED属性とUNBUFFERED属性

緩衝技法の属性は，RECORD属性をもつファイルに対してだけ適用される。BUFFERED属性は，ファイルとの間で転送されるレコードは中間的な主記憶域を通らなければならないことを示すものである。緩衝域の大きさは，そのファイルと結びつけられるデータ・セットの中のブロック（物理レコード）の大きさと，少くとも同じでなければならない。多くの場合，BUFFEREDが指定されているときは，データ転送と処理は，自動的に重複して行なわれる。

UNBUFFERED属性は，データ・セット内のレコードが緩衝域を通る必要はなく，その変数と結びつけられる主記憶域との間で直接，転送してよいことを示すものである。UNBUFFEREDが指定されているときは，データ転送と処理とが，自動的に重複して行なわれることはない。データ転送と処理を重複させたいときは，プログラマーがEVENTオプションを用いて行なわなければならない。

注：　UNBUFFEREDが指定されているときは，必らずといってよいほど，隠された緩衝域が使用されることになる。

付加的属性

付加的属性には，次のようなものがある。

PRINT
BACKWARDS
KEYED
ENVIRONMENT　（オプション・リスト）

PRINT属性

PRINT属性は，STREAM属性とOUTPUT属性をもつファイルに対してだけ適用される。これは，そのファイルが最終的には印刷されること。つまり，そのファイルと結びつけられるデータ・セットは，最初は何か別の媒体に書いてもよいが，最終的には印刷ページ上に出されることを示すものである。PRINT属性が与えられると，結びつけられたデータ・セットの各レコードの最初のバイトは，印刷制御文字としてとっておかれる。

BACKWARDS属性

BACKWARDS属性は，SEQUENTIAL，RECORD，および INPUT属性をもつファイルで，しかも磁気テープ上のデータ・セットであるものに対してだけ適用される。これは，ファイルが逆の順にアクセスされること。つまり，最後のレコードから始めて，最初のレコードがアクセスされるまで，進めてゆくことを示すものである。

KEYED属性

KEYED属性は，RECORD属性をもつファイルに対してだけ適用される。これは，データ転送ステートメントのキー・オプション（KEY，KEYTO，またはKEYFORM）のいずれか1つを用いて，ファイル内のレコードをアクセスすることができることを示すものである。KEYED属性は，実際にキーがデータ・セット上に存在していること。あるいはキーがデータ・セットに書かれたり，そこから読まれたりするということを，必らずしも示しているわけではないことに注意していただきたい。したがって，キー・オプションのいずれかが使用されていない限り，指定する必要はない。キーの性質および用法については，第12章「レコード方式の転送」のところでくわしく説明する。

ENVIRONMENT属性

ENVIRONMENT属性は，ファイルと結びつけられたデータを アクセスする方法をコンパイラーが決定するために必要な情報を与えるものである。この属性は，そのファイルと結びつけられるデー

タ・セットの物理的な構成を指定したり，データ・セットの処理方法を指示する。

ENVIRONMENT属性の一般形式は，次の通りである。

ENVIRONMENT （オプション・リスト）

ENVIRONMENT属性は，ファイル宣言の中で，あるいはCLOSE ステートメントのオプションとして与えることができる。CLOSE ステートメントの中でENVIRONMENT を指定するときに許されるオプションは，LEAVEまたはUNLOADだけである。

MEDIUMおよびFUNCTION オプションは別として，2つのタイプのデータ転送に特有のオプションについては，第11章「ストリーム方式の転送」および第12章「レコード方式の転送」のところで説明する。

装置タイプ(MEDIUM)： ENVIRONMENT属性のMEDIUMオプションは，VSAM ファイルの場合をのぞいて，ファイルの宣言の各々に対して指定しなければならない。VSAM の場合は，指定しても無視される。

MEDIUMオプションの形式は，次の通りである。

MEDIUM(記号装置名〔，物理装置タイプ〕)

記号装置名は，SYS×××の形式をしている。×××は，次のようにすることができる。

1. IPT……システム入力装置
2. LST……リスティング用に使用されるシステム出力装置
3. PCH……システム出力装置(カード穿孔装置)
4. 000から221……SYS000からSYS221までの記号装置名

装置タイプの指定には，使用する装置の番号を入れる。たとえば，IBM 1442−N1型 カード読取/穿孔装置を使用するときは，このオプションは，1442 と書く。図10.1は，個々の装置タイプの指定の仕方を示したものである。ある種の環境では，装置タイプの指定は任意選択であるが，MEDIUMオプションの中で指定した場合は，後で置き換えることはできない。このあとの「装置の独立性」の項を参照していただきたい。

装置タイプ	番号	装置タイプの指定
カード読取装置および穿孔装置	IBM 2540	2540
	IBM 2560	2560
	IBM 1442N1	1442
	IBM 1442N2	1442
	IBM 2520B1	2520
	IBM 2520B2	2520
	IBM 2520B3	2520
	IBM 2501	2501
	IBM 3504	3504
	IBM 3505	3505
	IBM 3525	
	(多重行印刷)	3525
	(2行印刷)	3525T
	IBM 3881	3881
	IBM 5425	5425
印刷装置	IBM 1403	1403
	IBM 1404	1404
	IBM 1443	1443
	IBM 1445	1445
	IBM 3211	3211
	IBM 5203	5203
	IBM 3203	3203
磁気テープ装置	IBM 2400 (9トラック)	2400
	IBM 2400 (7トラック)	2400
	IBM 3410/3411	3410
	IBM 3420	3420
DASD	IBM 2311	2311
	IBM 2314	2314
	IBM 2321	2321
	IBM 3330	3330
	IBM 3340	3340

図10.1 装置タイプと対応する指定

図10.1の中にリストされている装置タイプは，図10.2に示されているように，記号装置名SYSIPT，SYSLST，およびSYSPCHに割り当てることができる。

オペレーティング・システムの入出力機能は，オペレーティング・システムが使用する記号装置名とPL/I ファイルを結びつけるために，"記号装置名"を使用する。システムの中では，記号装置名は物理装置に，一時的または永久的に割り当てられる。しかし，連係編集プログラムによって，正しいデータ管理入出力ルーチンを目的プログラムに組み込むことができるように，コンパイラーは，PL/I ファイルと結びつけられるデータ・セットに対して用いられる物理装置のタイプを知っていなければならない。さらにくわしい点については，この章の後の方の「データ・セットとファイルの結びつけ方」のところをみていただきたい。物理装置タイプの選択は通常，コンパイレーションの前に行なわなければならないこと，およ

び変更をするときは通常，MEDIUMオプションを修正して，コンパイルし直すことになる点に注意していただきたい。しかし，ある程度，装置を独立させることは可能である。

装置の機能（FUNCTION）： ENVIRONMENT属性のFUNCTIONオプションは，IBM 3525，IBM 2560，またはIBM 5425によって実行される操作を指定するものである。IBM 3525のFUNCTIONオプションの形式は，次の通りである。

```
              ┌     ┐
              │,I   │
    ┌ R ┐     │,RP  │
FUNCTION ( { P }  { ,RW  } )
    └ W ┘     │,PW  │
              │,RPW │
              └     ┘
```

IBM 2560とIBM 5425のFUNCTIONオプションは，次の通りである。

```
           ┌ R ┐
FUNCTION ( { P } 〔1|2〕)
           └ W ┘
```

"R"，"P"および"W"という文字は，ファイルによって実行される操作を次のように指定するものである。

R　カードを読むためにファイルが使用される。ファイルは，INPUT 属性をもっていなければならない。

P　カードを穿孔するためにファイルが使用される。ファイルは，OUTPUT属性をもっていなければならないが，PRINT属性をもっていてはならない。

W　カード上に印刷するためにファイルが使用される。ファイルは，OUTPUT属性をもっていなければならない。IBM 3525 の場合だけは，PRINT属性をもっていてもよい。IBM 2560およびIBM 5425の場合は，印刷制御文字が使用できないので，ファイルは，PRINT属性も，CTLASAまたはCTL360オプションももつことはできない。

1および2という数字（IBM 2560およびIBM 5425の場合にだけ使用可能）は，ファイルの入力ホッパー選択を表わすものである。

FUNCTIONオプションの2番目のオペランド（IBM3525の場合にだけ使用可能）は，同じカードに対していくつかの操作をするためにこの装置を使用することを指定するものである。このオプションは，次のような値をもつことができる。

I　カードの穿孔および翻訳印刷のために，このファイルが使用される。この場合だけは，最初のオペランドを省略してもよい。指定するときは，"P"としなければならない。

RP　このファイルは，カードを読んでから，穿孔するために用いられる2つの関連ファイルの1つである。

RW　このファイルは，カードを読んでから，印刷するために用いられる2つの関連ファイルの1つである。

PW　このファイルは，カードを穿孔してから，印刷するために用いられる2つの関連ファイルの1つである。

RPW　このファイルは，カードを読み，穿孔し，次に印刷するために用いられる3つの関連ファイルの1つである。

IBM 3525に対するFUNCTIONオプションには，次の規則が適用される。

1. MEDIUMオプションの中で指定する装置タイプは，3525か3525Tでなければならない。
2. FUNCTIONオプションの2番目のオペランドが"I"のときは，第1番目のオペランド（指定するならば）は，"P"でなければならない。
3. COLBINオプションをFUNCTION（P，I）といっしょに指定してはならない。
4. FUNCTIONのオペランドが1つだけか，FUNCTION（P，I）でない限り，ASSOCIATEオプションを指定しなければならない。
5. FUNCTION（P，I）が指定されているときは，出力レコードはF形式でなければならない。

3525と結びつけられるファイルの宣言からFUNCTIONオプションが省略されているときは，次のような省略時解釈が適用される。

INPUTファイル	FUNCTION(R)
OUTPUT 非PRINTファイル	FUNCTION(P)
OUTPUT PRINTファイル	FUNCTION(W)

IBM 2560とIBM 5425の場合の省略時解釈は，次の通りである。

INPUTファイル	FUNCTION(R1)
OUTPUTファイル	FUNCTION(P1)

（ホッパー選択の数字だけが省略されている場合は，省略時解釈はホッパー1になる。）

装置の独立性

記号装置名SYSIPT，SYSLST，およびSYSPCHに対する記号装置名のMEDIUMオプションの中で，装置タイプを指定する必要はない。ユーザーは，プログラムの実行中に使用される実際の装置タイプを，ジョブ制御ASSGNステートメントを用いて指定することができる。（ただし，MEDIUMオプションの中で装置タイプが指定されているときは，後でそれを置き換えることはできない。）

記号装置名	装置タイプ
SYSIPT	IBM 2540 （読取装置） IBM 2560 （読取装置） IBM 1442N1 IBM 2501 IBM 2520B1 IBM 2400 （7または9トラック） IBM 2311 IBM 2314 IBM 3504 IBM 3505 IBM 3525 IBM 3420 IBM 3410／3411 IBM 3330 IBM 3340 IBM 3540 IBM 3881 IBM 5425
SYSLST	IBM 1403 IBM 1404 IBM 1443 IBM 2400 （7または9トラック） IBM 2311 IBM 2314 IBM 3211 IBM 3420 IBM 3410／3411 IBM 3330 IBM 3340 IBM 3540 IBM 3881 IBM 5203／3203
SYSPCH	IBM 2540 （穿孔装置） IBM 2560 （穿孔装置） IBM 1442N1 IBM 1442N2 IBM 2520B1 IBM 2520B2 IBM 2520B3 IBM 2400 （7または9トラック） IBM 2311 IBM 2314 IBM 3525 IBM 3410／3411 IBM 3420 IBM 3330 IBM 3340 IBM 3540 IBM 3881 IBM 5425

図10.2　SYSIPT，SYSLST，およびSYSPCHと結びつけられる装置のタイプ

次のような必要条件が満たされていれば，装置を独立させるために必要なテーブルとルーチンは，コンパイラーによって自動的に生成される。

1. 出力ファイルSYSLSTは，STREAM属性とPRINT属性をもっているか，またはCTLASAオプションをもつRECORD属性をもっていなければならない。出力ファイルSYSPCHは，CTLASAオプションをもつRECORD属性をもっていなければならない。このような場合は，各レコードの最初の文字は制御文字として解釈される。入力ファイル（SYSIPT）は，STREAMまたはRECORDのどちらでもよい。どの場合にも，RECORDファイルは，BUFFERED属性をもっていなければならない。

   注：　出力ファイルSYSPCHの場合に，ASSGNステートメントの中で指定された装置がIBM 2560のときは，装置の独立のファイル宣言でFUNCTIONオプションを指定することはできない。ホッパー選択は，ASSGNステートメントを通して行なわれ，入力と出力ファイルのそれぞれに対して読取と穿孔操作が行なわれる。IBM 2560を使用する出力用の装置独立のファイルの場合は，"V"および"W"ANS 制御文字しか使用することはできない。

2. IBM 5425多能カード装置は，96桁カードを使用する。SYSIPTまたはSYSPCHに割り当てられているときは，80桁のデータだけを読んだり，穿孔したりすることができる。96桁全体を読んだり，穿孔したりするためには，MEDIUMオプションで5425を指定しなければならない。

3. レコードは，非ブロック化，固定長のものでなければならない。レコード・サイズは，次の通りでなければならない。

   SYSIPTの場合は，≦ 80
   SYSLSTの場合は，≦121
   SYSPCHの場合は，≦ 81

   注：　データ・セットが直接アクセス装置上にあるときは，SYSIPT，SYSLST，およびSYSPCHの値はそれぞれ，レコード・サイズ80，121，および81でなければならない。

上の条件にあっているときは，それぞれの記号装置名に対してサポートされる装置タイプ（図10.2を参照）はいずれも，コンパイルし直す必要なしに，割り当てることができる。

ファイルの開閉

ファイルに結びつけられたデータを転送できるようにするためには，その前に，外部記憶媒体の使用可能性のチェック，媒体の位置づけ，該当するオペレーティング・システムのサポートの割振り，などのファイル準備作業が行なわれていなければならない。このような作業を，ファイルを開く，という。また，処理が終ったときは，ファイルを閉じなければならない。ファイルを閉じるということは，ファイルを開くときに設定された機能を解放することを意味している。

PL/Iには，このような機能を行なうためのステートメントが2つ-OPENとCLOSE-ある。しかし，これらのステートメントは，任意選択である。ファイルに対してOPENステートメントが実行されなかった場合は，そのファイルに対する最初のデータ転送ステートメントが実行される前に，ファイルは自動的に開かれる。この場合の自動的なファイルの準備は，DECLAREステートメントで指定されたファイルに関する情報を用いる，いくつかの標準システム・プロシージャーによって行なわれる。同様に，プログラムが完了する前にファイルが閉じられなかった場合は，プログラムが正常に完了するときに，ファイルは自動的に閉じられる。

OPENステートメント

OPENステートメントが実行されると，1つまたは2つ以上のファイルが明示的に開かれる。OPENステートメントの基本形式は，次の通りである。

```
OPEN FILE(ファイル式)〔オプション・グループ〕
〔,FILE(ファイル式)
〔オプション・グループ〕〕…;
```

OPENステートメントのオプション・グループでは，CONSECUTIVE編成のUNBUFFEREDファイルに対するINPUTまたはOUTPUT属性，TITLEオプション，およびPAGESIZEとLINESIZEオプションを指定することができる。(PAGESIZEとLINESIZEオプションは，STREAM OUTPUTファイルにだけ適用される。)OPENステートメントの中の属性とオプションは，DECLAREステートメントの中で記述されたものと組み合わせられる。同じファイルに対するOPENステートメントとDECLAREステートメントの両方で，属性を繰返して指定する必要はない。2組の属性の間に矛盾があると，UNDEFINED条件が起こる。

ファイルを明示的に開く場合は，そのファイルに対する入出力ステートメントが実行される前に，OPENステートメントを実行しなければならない。

暗示的な開き方

開かれていないファイルに対して入出力ステートメントが実行されると，ファイルは暗示的に開かれる。

データ・セットとファイルの結びつけ方

PL/Iプログラムの中のファイルは，通常，ENVIRONMENT属性のオプションを用いて，物理的なデータ・セットに結びつけられる。どのようなファイル宣言の中でも，MEDIUMオプションを指定しなければならない。MEDIUMオプションでは，そのデータ・セットに対して，必ず'記号装置名'と，任意選択で'物理装置名'を指定しなければならない。'記号装置名'は，特定の入出力装置を識別するために，オペレーティング・システムが使用するものである。この形式は，'SYS×××'の形をとり，×××は，000から221まで，または，標準システム装置の場合は，SYSRDR，SYSIPT，SYSIN，SYSOUT，SYSLST，SYSLNK，およびSYSLOGである。特定のシステムの場合，記号装置名は，特定の物理装置に割り当てられる。たゞし，これらの割当ては，ジョブ制御ASSGNステートメントによって一時的に変更することができる。本章の「装置の独立性」の項で前に述べたような特殊な場合をのぞいて，物理装置タイプは，原始プログラムの中で指定し，コンパイラーが，該当のデータ管理ルーチンを連係編集することができるようにしなければならない。

MEDIUMオプションの物理装置タイプは，自動的か，またはASSGNステートメントを用いてファイルに割り当てられた物理入出力装置タイプに対応するものでなければならない。たとえば，物理装置タイプが磁気テープを表示している場合は，ファイルは，磁気テープ装置に割り当てなければならない。

次の例について考えてみよう。

```
DECLARE MASTER FILE RECORD INPUT
        SEQUENTIAL ENVIRONMENT
(…MEDIUM(SYS006, 2400)… );
```

この宣言では，ファイルMASTERは，記号装置名SYS006に割り当てられており，MASTERの物理装置タイプは，IBM 2400磁気テープ装置であると宣言されている。

上記のファイル宣言が使用されている場合に，オペレーティング・システムが，自動的にSYS006を適当な磁気テープ装置に結びつけるときは，それ以上に割当てをする必要はない。そうでないときは，ASSGNステートメントを用いて，システム・ファイルを磁気テープ装置に割り当てなければならない。その実行モジュールといっしょに使用されるジョブ制御プログラムには，次のようなステートメント(一部分だけを示す)が含まれていなければならない。

```
// ASSGN SYS006, …
```

このステートメントは，論理装置名SYS006を物理入出力装置(もちろん，磁気テープでなければならない)に結びつけるものである。使用される所定のテープ装置を，ASSGNステートメントのSYS006の後に書く。

MEDIUMオプションは，ファイルを装置またはボリュームに結びつけるだけである。TITLEオプションの中で指定されたファイル名または識別名の最初の7文字によって，ファイルは，ジョブ

制御ステートメントDLBL またはTLBL に結びつけられる。

OPEN ステートメントの中で使用されているファイル式がファイル定数の場合は，TLBL またはDLBL ステートメントで，ファイル定数を指名しなければならない。OPEN ステートメントでファイル定数を指定していない場合は，TLBL またはDLBL スステートメントで，ファイル式の値を指名しなければならない。たとえば，

```
PRICES = RPRICE;
OPEN FILE(PRICES);
```

TLBL ステートメントは，次のように，データ・セットSTOCK をファイル定数RPRICE に結びつける。

```
// TLBL RPRICE, 'STOCK'
```

TITLE オプションを含むOPENステートメントによってファイルが開かれる場合は，そのプログラムの外では，そのファイル名は使用されない。TITLEオプションの中の式は評価され，必要なら，適当なデータ・セットを識別するものとみなされる文字ストリングに変換される。次にあげるOPEN ステートメントは，TITLE オプションの用い方を示したものである。

```
OPEN FILE(DETAIL) TITLE('DETAIL1');
```

このステートメントを実行するときは，DETAIL1をそのファイル名としてもつTLBL またはDLBL ステートメントが，現行のジョブ・ステップの中に入っていなければならない。そのステートメントは，次のようなものである。

```
// TLBL DETAIL1, 'DETAILA'
```

したがって，データ・セットDETAILAは，TLBLファイル名，DETAIL1を通して，ファイルDETAIL にに結びつける。

TITLEオプションを使用すると，プログラマーは，いくつかのデータ・セットの中から，特定のファイル名と結びつけられるものを，OPENのときに動的に選ぶことができる。たとえば，

```
DECLARE 1  INREC,
             2  FIELD_1…,
             2  FILE_ID CHAR(7),
        DTL FILE INPUT…,
        MSTR FILE INPUT…;
OPEN FILE(DTL);
READ FILE(DTL) INTO(INREC);
OPEN FILE(MSTR) TITLE(FILE_ID)
```

これらのステートメントが含まれているプログラムは，数個の異なる明細データ・セットを処理するためのものであり，それらのデータ・セットに対応するマスター・データ・セットはそれぞれ異なっているとしよう。さらに，各々の明細データ・セットの最初のレコードには，最後のデータ項目として，該当するマスター・データ・セットを識別する文字ストリングが含まれているとしよう。次のようなジョブ制御ステートメントが，現行のジョブ・ステップの中にあるはずである。

```
// DLBL DTL, '…
// DLBL MSTR1A,  'MSTR1A'
// DLBL MSTR1B,  'MSTR1B'
// DLBL MSTR1C,  'MSTR1C'
```

この場合，MSTR1A，MSTR1B，およびMSTR1Cは，3つの異なるマスター・ファイルを表わしている。DTLの最初のレコードには，最後のデータ項目として，'MSTR1A'，'MSTR1B'，または'MSTR1C'が含まれており，これが文字ストリング変数FILE_IDに割り当てられる。ファイルMSTRが開かれると，FILE_IDの現在の値によって，該当するデータ・セットが識別される。

ファイル変数を使用しても，1つのステートメントによって様々な時点にいくつものファイルを操作することができる。たとえば，

```
DECLARE  F FILE VARIABLE,
         A FILE……,
         B FILE……,
         C FILE……;
           ⋮
         F=A;
           ⋮
LAB:     READ FILE (F)……;
           ⋮
         F=B;
         GO TO LAB;
           ⋮
         F=C;
         GO TO LAB;
```

LABというラベルのついたステートメントは，A，B，Cの3つのファイルを読むために使用されている。各々の場合にREADステートメントが実行されたあとも，ファイルA，B，Cはいずれも開かれたままであることに注意していただきたい。1つのステートメントでいくつものデータ・セットをアクセスできるようにするためにTITLEオプションが使用されていたとすれば，ファイルを別のデータ・セットに結びつける必要があるたびに，ファイルを閉じて，開き直さなければならない。1つのステートメントによっていくつものデータ・セットをアクセスするときは，TITLEオプションよりもファイル変数を使用した方が，プログラムに必要な限り各データ・セットに対するファイルを開いたままにしておくことができるので，プログラムの効率がよい。

CLOSEステートメント

CLOSE ステートメントの基本的な形式は次の通りである。

```
CLOSE FILE(ファイル式)
  〔ENVIRONMENT (LEAVE|UNLOAD)〕
     〔,FILE(ファイル式)
       〔ENVIRONMENT (LEAVE|UNLOAD)〕〕…;
```

CLOSEステートメントが実行されると，指定されたファイルは，そのファイルが開かれたときに結びつけられたデータ・セットから切り離される。また，CLOSEステートメントは，暗示的または明示的にファイルが開かれるときに与えられたすべての属性を，ファイルから切り離す働きもする。必要ならば，後続のOPENステートメントの中で，そのファイル定数に対して新しい属性を指定することができる。しかし，DECLAREステートメントの中でファイル定数に対して明示的に与えられた属性はすべて，効力をもちつづける。

OPENステートメントの場合と同様，1つのCLOSEステートメントで2つ以上のファイルを閉じるようにすると，実行時間を減らすことはできるが，1つずつ閉じた場合にくらべて，一時的に使用される内部記憶域の量は増すことになる。

LEAVEおよびUNLOADオプションは，ファイルを閉じるときの磁気テープの後始末を制御するためのものである。

注：すでに閉じられているファイルを閉じたり，すでに開かれているファイルを開いても，プログラムの実行時間が増すだけで，何の効果もない。

標準ファイル

PL/Iプログラムで使用できる標準ファイルが，2つある。1つは，標準入力ファイルSYSINで，もう1つは標準出力ファイルSYSPRINTである。これらのファイルは，明示的に宣言したり，開いたりする必要がなく，標準的な属性のセットが自動的に適用される。SYSINの属性は，STREAM INPUTで，SYSPRINTの属性は，STREAM OUTPUT PRINTである。双方のファイル名SYSINとSYSPRINTは，EXTERNAL属性をもつものとみなされる(SYSPRINTの桁数は7桁より大きいが)。

これらのファイルを使用するときに，GETおよびPUTステートメントの中でFILEオプションを指定する必要はない。ファイルを指名していないGETおよびPUTステートメントは，次のものと同じになる。

```
GET FILE(SYSIN)…;
PUT FILE(SYSPRINT)…;
```

上記以外にSYSINおよびSYSOUTを参照するとき(たとえば，ONステートメントの中，またはレコード方式のステートメントの中で)は，明示的に記述しなければならない。

SYSINおよびSYSPRINTという識別名は，上述のような特別の目的に制限されているわけではなく，標準ファイルの識別以外の目的にも使用することができる。その他の属性を，明示または文脈のどちらかによってこれらに適用することもできるが，SYSPRINTがSTREAM OUTPUTファイルとして宣言されたり，開かれたりしている場合は，INTERNAL属性が宣言されていない限り，PRINT属性が適用される。

注：SYSINまたはSYSPRINTがSTREAMファイル以外のものとして宣言されている場合は，特別に注意しなければならない。実際に，コンパイラーは，ファイル定数が明示的に記述されていないGETステートメントの各々に識別名SYSINを挿入し，ファイル定数が明示的に記述されていないPUTステートメントの各々に識別名SYSPRINTを挿入する。したがって，次の例はエラーになる。

```
DECLARE (SYSIN, SYSPRINT) FIXED
      DECIMAL (4,2);
         ⋮
GET LIST (A,B,C);
PUT LIST (D,E,F);
```

識別名SYSINがGETステートメントに挿入され，SYSPRINTがPUTステートメントに挿入される。しかし，この場合，これらは，標準ファイルを指しているのではなく，そのブロックの中で宣言されている固定小数点変数を指すことになる。

第11章： ストリーム方式の転送

この章では，ストリーム方式の転送で用いられる入出力ステートメントについて解説する。ストリーム方式転送とレコード方式転送の両方に同じように適用される機能（ファイル，ファイルの属性，ファイルの開閉など）については，第10章「入力および出力」で説明した。

ストリーム方式の転送では，データ・セットは，文字形式をしたデータ項目の連続する流れ（ストリーム）として扱われ，プログラムの中では，ブロックおよびレコードの境界は無視される。しかし，データ・セットは，一連のデータの行からなると考えられ，ストリーム方式転送によって作成またはアクセスされるデータ・セットの各々には，行サイズが関連づけられている。通常，行は，データ・セット内のレコードと同じであるが，行サイズは，レコード・サイズと必ずしも同じにしなくてもよい。

ストリーム方式の転送には，3つのモード - リスト指示，データ指示，および編集指示 - がある。3つのモードのいずれで使用される転送ステートメントにも，次の情報が必要である。

1. データを取り出すデータ・セット，またはデータを割り当てるデータ・セットと結びつけられるファイルの名前。

2. 入力中にデータ項目が割り当てられたり，出力中にデータ項目が得られる，プログラム変数のリスト。このリストは，データ・リストと呼ばれる。リスト指示，および編集指示モードの出力の場合は，データ・リストに式が含まれていてもよい。

3. 編集指示モードの場合は，ストリームの中にある各データ項目の形式。

特定の状況のもとでは，この必要情報のいくつかは，暗示することができる。

リスト指示転送

リスト指示転送の場合は，ユーザーは，ストリーム内における項目の形式を指定せずに，データを読み込んだり，書き出したりすることができる。

入力： 通常，ストリーム内のデータ項目は，符号付（任意選択）の有効な定数の形をした文字ストリングか，または複素数定数を表わす式の形をした文字ストリングである。データ項目を割り当てる変数は，データ・リストによって指定する。各項目は，カンマおよび／または1個以上のブランクによって区切る。

出力： 転送されるデータ値は，変数，定数，またはデータ項目を表わす式によって指定する。ストリームに入れられる各データ項目は，変数の属性を反映した文字ストリング表現である。各項目は，1個以上のブランクによって区切る。算術データの先行ゼロは，抑止される。2進数項目は，10進数表示で表わされる。

PRINTファイルの場合，各データ項目は，そのコンパイラーによって前もって決められているタブ位置に，自動的に合わせられる。これらの位置は，1，25，49，73，97，121であるが，プログラマーがこれらの値を変更できるような手段も用意されている。

データ指示転送

データ指示転送を使用した場合は，ユーザーは，自己定義のデータを転送することができる。

入力： ストリームの中の各データ項目は，それが割り当てられる値と変数の両方を指定する割当ての形式をしている。通常，値は定数の形になっている。各項目は，カンマおよび／または1つ以上のブランクによって区切る。1つのGETステートメントによってアクセスされる項目グループの最後にセミコロンをつけなければならない。ストリームから取り出される項目の数はセミコロンによって決まるので，GETステートメントの中のデータ・リストは任意選択である。

出力： 転送されるデータの値は，任意選択のデータ・リストによって指定することができる。ストリームの中に入れられる各データ項目は，セミコロンをもたない割当てステートメントの形式にする。各項目は，1つ以上のブランクで区切る。各々のPUTステートメントによって転送される最後の項目の後には，セミコロンをつける。算術データの先行ゼロは，抑止される。各々の値の文字表現は，変数の属性を反映したものになるが，2進数項目の場合は例外で，10進数表示で表わされた値になる。

データ・リストが省略されているときは，そのステートメントが含まれているブロックの中で知られており，しかもデータ指示出力の中で許されるすべての変数を指定したものとみなされる。

PRINT ファイルの場合は，データ項目は，そのコンパイラーで定義されたタブ位置に自動的に合わせられる。これらの位置は，1，25，49，73，97および121であるが，プログラマーがこれらの値を変更できるような手段も用意されている。

編集指示転送

編集指示転送では，ユーザーは，データが割り当てられる変数を指定するか，または転送されるデータを指定し，さらに外部媒体上における各項目の形式を指定することができる。

入力： ストリーム内のデータは，連続する文字のストリングであり，データ項目を別々に分離することはしない。データが割り当てられる変数は，データ・リストによって指定する。形式リストの中の形式項目では，各変数に割り当てられる値が含まれている文字の桁数を指定し，そのデータの性質（たとえば，仮想上の小数点の位置など）を記述する。

出力： 転送されるデータ値は，データ・リストによって定義する。ストリーム内におけるデータの形式は，形式リストによって定義する。

データ転送ステートメント

ストリーム方式の転送では，入力ステートメントとしてはGETだけ，出力ステートメントとしてはPUTだけを使用する。GETステートメントは，ストリームから次の一連のデータ項目を取り出す働きをもち，PUTステートメントは，指定された一組のデータ項目をストリームに入れる働きをする。データ項目が割り当てられる変数，およびそこからデータが転送される変数または式は通常，各々のGETまたはPUTステートメントのデータ・リストの中で指定する。このステートメントには，データの出所または行先を指定するオプション，または先行データからみて相対的にストリーム内のどこにそのデータがあるかを表示するオプションも含めることができる。

次にあげるのは，ストリーム方式のデータ転送ステートメントとそのオプションの要約である。

STREAM INPUT：

```
GET 〔FILE （ファイル式） | STRING
      （文字ストリング変数）〕
      〔データ指定〕
      〔COPY〔（ファイル式）〕〕
      〔SKIP〔（式）〕〕;
```

GETステートメントのSTRINGオプションでは，COPYオプションもSKIPオプションも使用することはできない点に注意していただきたい。

STREAM OUTPUT：

```
PUT 〔FILE （ファイル式） | STRING
      （文字ストリング変数）〕
      〔データ指定〕
      〔SKIP 〔（式）〕〕;
```

PUTステートメントの中で，SKIPオプションとSTRINGオプションをいっしょに用いることはできない点に注意していただきたい。

STREAM OUTPUT PRINT：

```
PUT 〔FILE （ファイル式）〕
      〔データ指定〕
        ⎧PAGE〔LINE（式）〕⎫
        ⎨SKIP〔（式）〕     ⎬
        ⎩LINE （式）       ⎭ ;
```

このオプションは，どのような順に書いてもよい。データ指定は，次の形式のいずれかにすることができる。

```
〔LIST〕（データ・リスト）
DATA 〔（データ・リスト）〕
EDIT {（データ・リスト） （形式リスト）}…
```

キーワードLIST，DATA，またはEDITのいずれかが前についていないデータ・リストが，GETまたはPUTステートメントに含まれているときは，LISTとみなされる。このようなステートメントの中では，GETまたはPUTというキーワードのすぐ後にデータ・リストを書かなければならない。必要なオプションがあれば，データ・リストの後で指定しなければならない。

制御オプション（PAGE，SKIPまたはLINE）のいずれかがあるときにだけ，データ指定を省略することができる。形式リストは，以下の形式項目のいずれかを使用することができる。

A,B,C,E,F, P,R,X	STREAMファイルまたはFILEオプションといっしょに指定することができる。
SKIP〔(W)〕 COLUMN (W)	どのようなSTREAMファイルの場合も指定することができる。
PAGE LINE (W)	STREAM OUTPUT PRINTファイルの場合に使用することができる。

これらのステートメントについては，J節「ステートメント」のところで，個々にくわしく説明する。

転送ステートメントのオプション

FILEおよびSTRINGオプション

FILEオプションは，それに対して操作が行なわれるファイルを

指定するものである。STRINGオプションを使用した場合は，内部記憶装置と外部記憶装置の間ではなく，内部記憶装置同志の間でデータを転送するためにGETおよびPUTステートメントを使用することができる。GETステートメントの中にFILEオプションもSTRINGオプションもない場合には，標準入力ファイルSYSINとみなされる。PUTステートメントの中にどちらのオプションもない場合は，標準出力ファイルSYSPRINTとみなされる。

以下にあげるステートメントのいくつかの中に，FILEオプションの使用法の例が出ている。第13章「編集およびストリングの取扱い」のところに，STRINGオプションの用法例があげてある。

COPYオプション

COPYオプションは，GET FILEステートメントの中でだけ使用することができる。このオプションは，読まれた通りに，ストリームを指定のファイルに書き出すことを指定するものである。ファイルが指定されていない場合の省略時解釈は，標準出力ファイルSYSPRINTである。たとえば，次のステートメントは

```
GET FILE(SYSIN) DATA(A,B,C) COPY(DPL);
```

入力ストリームの中のA，B，Cに割り当てられた値をこれらの名前をもつ変数に転送するばかりでなく，それらが入力ストリームの中にあった通りの形でファイルDPLに書き出す働きもする。これらが省略時解釈によってSYSPRINTファイルに書き出される場合は，データ指示形式になる。入力でスキップされるデータ項目および内部変数に転送されないデータ項目は，もとのままで出力ストリームに複写される。

SKIPオプション

SKIPオプションは，データ・セットの中の新しい現在の行（またはレコード）を指定するものである。括弧内の式は，整数wに変換される。データ・セットは，現在の行（レコード）から相対的にみてw番目の行（レコード）の先頭に位置づけされる。

PRINT以外のファイルの場合に，式が省略されているか，またはwがゼロより大きくないときは，値は1とみなされる。PRINTファイルの場合に，式が省略されているときは，値は1とみなされる。wがゼロ以下の場合は，現在と同じ行にキャリッジの戻りが行なわれる。

SKIPオプションは，それがデータ指定の後に書かれていても，データ指定によって定義された値が転送される前に，実行される。したがって，次のステートメントでは，

例：

```
PUT LIST(X,Y,Z) SKIP(3);
```

変数X，Y，Zの値は，現在の行より3行後の行から，標準出力ファイルSYSPRINT上に印刷される。

PAGEオプション

PAGEオプションは，PRINTファイルの場合にだけ指定することができる。これは，データ・セットの中に，新しい現在ページを定義するものである。PAGEオプションは，それがデータ指定の後に書かれていても，データ指定（あれば）によって定義されている値が転送される前に実行される。

LINEオプション

LINEオプションは，PRINTファイルの場合にだけ指定することができる。このオプションは，次の行が現在のページのw番目の行になるように，ブランクの行を挿入する働きをする（このwは，整数に変換されたときの括弧に入れた式の値である）。LINEオプションは，データ指定の後にあっても，データ指定によって定義された値が転送される前に実行される。PAGEオプションとLINEオプションの両方が同じステートメントにある場合は，PAGEオプションが最初に適用される。たとえば，次のステートメント

```
PUT FILE(LIST) DATA(P,Q,R) LINE(34) PAGE;
```

が与えられると，変数P，Q，Rの値が，新しいページの34行目からデータ指示形式で印刷される。

データ指定

転送されるデータを識別するために，GETおよびPUTステートメントの中でデータ指定が与えられる。データ指定は，転送のモードに対応している。

データ・リスト

リスト指示および編集指示のデータ指定は，転送されるデータ項目を指定するためのデータ・リストを必要とする。データ指示のデータ指定の場合は，データ・リストは任意選択である。

一般形式：

（データ・リスト）

ここで，"データ・リスト"は，次のように定義される。

要素 〔，要素〕…

統語規則：

要素の性質は，データ・リストが入力または出力のどちらに使用されているかによって異なる。その規則は次の通りである。

1. 入力の場合，編集指示およびリスト指示転送のデータ・リストの要素は，要素変数，配列変数，構造変数，STRING以外の擬似変数，またはこれらの要素のいずれかを含む繰返し指定（DOグループの繰返し指定と同じ）のいずれかにすることができる。データ指示のデータ指定の場合は，データ・リストの要素は，要素，配列または構造変数にすることができる。データ指示のデータ・リストの中の名前は，添字をつけたり，iSUB定義にすることはできないが，修飾された名前，単純定義された名前，またはストリング・オーバーレー定義の名前は許される。

2. 出力の場合，編集指示およびリスト指示のデータ指定のデータ・リスト要素は，要素式，配列式，構造式，またはこれらの要素のいずれかを含む繰返し指定のいずれかにすることができる。データ指示のデータ指定の場合は，データ・リスト要素は，要素変数，配列変数，構造変数，またはこれらの要素のいずれかを含む繰返し指定のいずれかにすることができる。これは，iSUB定義にすることはできないが，単純またはストリング・オーバーレー定義にすることはできる。データ指示出力の場合は，添字をつけてよい。

3. データ・リストの要素は，次のものであってよい。

   入力： 問題データ： 算術ストリング

   出力： 問題データ： 算術ストリング

   プログラム制御データ：
   エリア
   入口
   事象
   ファイル
   ラベル
   オフセット
   ポインター

   プログラム制御データを指定するデータ・リストは，PUT DATAステートメントの中でだけ使用することができ，変数の名前だけが転送される。

4. データ・リストはつねに，括弧で囲まなければならない。

繰返し指定

繰返し指定の一般形式が，図11.1に示してある。

統語規則：

1. 繰返し指定の要素リストの中の要素は，上にリストしたように，データ・リスト要素として許されるものであれば，いずれであってもよい。

2. この指定の中の式は，DOステートメントの中の指定と同じであり，次のように記述されている。

   a. 指定の中の各々の式は，要素式である。

   b. この指定の中では，式・1は，制御変数または擬似変数の最初の値を表わしている。式・3は，繰返し指定の中でデータ・リスト要素を繰り返すたびにそのあとで制御変数に加えられる増分を表わしている。式・2は，制御変数の最終値を表わしている。式・4は，繰返しの回数を制御する第2の条件を表わしている。この指定の正確な意味は，同じ指定をもつDOステートメントの意味と同じである。最後の指定が完了すると，データ・リスト内の次の要素に制御が渡される。

3. 一般形式のところに示してあるように，繰返しの指定は各々括弧で囲まなければならない。繰返し指定が，データ・リストの中の唯一の要素である場合は，データ・リスト用に1組の括弧，繰返し指定用にもう1組の括弧が必要なので，外側の括弧は2組必要である。

4. 図11.1に示すように，繰返し指定の"指定"の部分は，次の形式のようにして何回も繰り返すことができる。

```
DO I = 1 TO 4, 6 TO 10;
```

```
（要素 〔，要素〕…DO ｛変数    ｝ ＝ 指定〔，指定〕…）
                     ｛擬似変数｝
```

"指定"は，次のような形式をしている。

```
式－1 ［TO 式－2〔BY 式－3〕］ 〔WHILE （式－4）〕
      ［BY 式－3〔TO 式－2〕］
```

図11.1　繰返し指定の一般形式

繰返し指定は，ネストすることができる。つまり，繰返し指定の要素自体を繰返し指定にしてもよいということである。DO部分はそれぞれ，右側で右括弧を用いて区切らなければならない（それに対応する左括弧は，繰返し指定全体の始めに加える）。

DO部分がネストされているときは，ネスティングの外側のレベルにあるのは，右端のDOである。たとえば，次のステートメントについて考えてみよう。

```
GET LIST (((A(I,J) DO I = 1 TO 2)
           DO J = 3 TO 4));
```

添字を区切るのに用いられている1組の括弧のほかに，3組の括弧があることに注意していただきたい。1番外側の1組は，データ・リストに必要なものである。次の1組は，外側の繰返し指定に必要なものである。3番目の括弧は，内側の繰返し指定に必要なものである。このステートメントは，次のようなネストされたDO－グループと同じである。

```
DO J = 3 TO 4;
    DO I = 1 TO 2;
    GET LIST (A (I,J));
    END;
END;
```

これは，配列Aの要素に次のような順序で値を与えるものである。

```
A(1,3), A(2,3), A(1,4), A(2,4)
```

ネスティングで許される最大のレベルは，50である。

注：　DOキーワードは，繰返し指定の中で使用することができるが，対応するENDステートメントは，使用できない。

データ・リスト要素の転送

データ・リスト要素が複素数モードのものである場合は，実数部分が，虚数部分の前に転送される。

データ・リスト要素が配列変数の場合は，配列内の各要素は，行を主とした順序で－つまり，配列の右端の添字が最も頻繁に変わる形で－転送される。

データ・リスト要素が構造変数の場合は，構造内の各要素は，構造の宣言の中で指定された順に転送される。たとえば，宣言が次のようなものであるとすれば，

```
DECLARE 1 A (10), 2 B, 2 C;
```

次のステートメントは，

```
PUT FILE(X) LIST(A);
```

次のような順序で出力されることになる。

```
A.B(1) A.C(1) A.B(2) A.C(2) A.B(3)
 A.C(3)…
```

しかし，宣言が次のようなものであったとすれば，

```
DECLARE 1 A, 2 B(10), 2 C(10);
```

同じPUTステートメントを与えても，次のような順序で出力されることになる。

```
A.B(1) A.B(2) A.B(3)…A.B(10)
A.C(1) A.C(2) A.C(3)…A.C(10)
```

リスト指示または編集指示転送の入力ステートメントで用いられるデータ・リストの中で，変数に値が割り当てられるときは，データ・リストの中でのちに参照が行なわれたときにこの変数があれば，この新しい値が用いられる。たとえば，

```
GET LIST (N,(X(I) DO I=1 TO N), J, K,
  SUBSTR (NAME, J, K));
```

このステートメントが実行されると，データは，次のような順序で転送され，割り当てられる。

1. 新しい値がNに割り当てられる。

2. 繰返し指定の中で指定されているようにX(1), X(2), …X(N)の順序で，要素が配列Xに割り当てられる。このとき，割り当てられる項目の数を指定するために，新しいNの値が用いられる。

3. 新しい値がJに割り当てられる。

4. 新しい値がKに割り当てられる。

5. 長さKのサブストリングが，ストリング変数NAMEにJ番目の桁から割り当てられる。

リスト指示のデータ指定

リスト指示のデータ指定の一般形式は，入力のときも出力のときも，次の通りである。

〔LIST〕(データ・リスト)

データ・リストは，前記の「データ・リスト」のところで述べたものと同じである。しかし，データ・リストがGETまたはPUTキーワードのすぐ後にある場合だけは，省略時解釈によって想定することができる。

次にあげるのは，リスト指示のデータ指定の例である。

```
LIST (CARD, RATE, DYNAMIC_FLOW)
LIST ((THICKNESS(DISTANCE)
  DO DISTANCE = 1 TO 1000))
LIST (P, Z, M, R)
LIST (A*B/C, (X+Y)**2)
```

最後の例の指定は，式によって指定された値をもっているので，出力の場合にだけ使用することができる。このような式は，ステートメントが実行されて，結果がストリームに入れられるときに評価される。

ストリームの中のリスト指示データ

ストリームの中にある問題データは，入力の場合も出力の場合も，文字データ・タイプであり，次のような一般形式のいずれかをとる。

〔+1-〕　算術定数

文字ストリング定数

ビット・ストリング定数

〔+1-〕　実数定数{+1-}虚数定数

ストリング定数は，上にリストした2つの形式のどちらかでなければならず，反復とストリング繰返し因数は使用できない。実数定数の前にある符号のあとにブランクをつけてはならないし，また，複素数式の中央にある+または-の前後にブランクがあってはならない。

リスト指示の入力形式

指名されたデータが配列のときは，データは定数により構成され，その最初の定数は配列の最初の要素に，2番目の定数は2番目の要素に，というように，行を主とした順序で割り当てられる。

データ・リストの中の構造名は，その中に含まれる要素変数と配列のリスト(構造の記述の中で指定された順序で並べられている)を表わしている。

入力の場合，ストリームの中のデータ項目は，ブランクかカンマで区切らなければならない。この区切りの前後には，任意の数のブランクがあってよい。ストリーム内のナル・フィールドは，データ・ストリームの中にある最初の非ブランク文字(カンマ)か，または任意の個数のブランクで区切られた2つのカンマ，のどちらかによって表示される。ナル・フィールドは，データ・リストの中の関連する項目の値を変えずにおくことを指定するものである。

入力における，定数のリストの転送は，リストがなくなったとき，またはファイルの終りになったときに終了する。前者の場合，ファイルは次のGETステートメントが実行できる位置に，ストリームの中で位置づけされている。2つ以上のブランクをあけて2つのデータ項目を区切ることができるし，また，1つのカンマ区切記号の前または後に1つ以上のブランクをおくこともできる。

各項目がカンマで区切られているときは，次のGETステートメントが実行されるときに走査される最初の文字は，カンマのすぐ後にあるものになる。各項目がブランクだけで区切られているときは，走査される最初の項目は，次の非ブランク文字になる。下の例の中で，Xは非ブランク文字，bはブランクを表わし，↑は次のGETステートメントが開始されるときのファイルの位置を示している。

```
Xbb,bbbXXX
    ↑
XbbbbXXXX
     ↑
Xb,]←レコードの終り
   ↑
```

レコードがカンマで終っているときは，次のGETステートメント

で必要になるまで，後続のレコードは読み込まれないことに注意していただきたい。

データが文字ストリング定数の場合は，前後の引用符号は除去され，その中にあった文字が文字ストリングとして解釈される。

データがビット・ストリング定数の場合は，前後の引用符号と後につけられた文字Bが除去され，その中にあった文字がビット・ストリングとして解釈される。

データが算術定数または複素数式の場合は，定数によって暗示される基数，スケール，モード，および精度をもつコード化算術データとして解釈される。

リスト指示の出力形式

データ・リストの中の要素変数と式の値は，文字表現に変換され，データ・ストリームに転送される。

転送される，連続したデータ項目は，ブランクで区切る。(PRINTファイルの場合は，各項目は，プログラムのタブのセット位置にしたがって区切られる。)

ストリームの中に入れられるデータ・フィールドの長さは，データ項目の属性(精度および長さを含めて)によって決まる。変換の規則と，精度に及ぼす影響については，F節「データの変換と式の値の計算」の中の，文字タイプへの変換のところにリストされている。

2進数データ項目は，ストリームに入れられる前に10進数表現に変換される。

数字の値の場合は，文字ストリングの値が転送される。

ビット・ストリングは，ビット・ストリング定数(文字0と1からなり，引用符で囲み，さらにその後に文字Bをつけたもの)の文字表現に変換される。

文字ストリングは，次のように書き出される。ファイルがPRINT属性をもっていないときは，前後に引用符が補なわれ，その中にある単一引用符またはアポストロフィーは，2つの引用符で置き換えられる。フィールドの幅は，ストリングの現在の長さと，追加された引用符の数を加えたものになる。ファイルがPRINT 属性をもっているときは，前後の引用符は補なわれず，その中にある単一引用符も修正されない。フィールドの幅は，ストリングの現在の長さになる。

データ指示のデータ表示

データ指示のデータ指定の一般形式は，入力の場合も出力の場合も，次の通りである。

DATA〔(データ・リスト)〕

一般的な規則：

1. データ・リストについては，この章の「データ・リスト」のところで説明した通りである。入力の場合は，データ・リストに添字付の名前を含めることはできない。データ・リストの中の構造要素の名前は，不明確にならない程度に修飾するだけでよく，完全に修飾する必要はない。入力の場合，認識不可能な要素変数，またはデータ・リストの中に相手をもたない名前がストリームに含まれているときは，NAME 条件が起こる。

2. データ・リストを省略することは，データ・リストがあると想定されることを意味している。想定されたこのデータ・リストには，そのブロックまたはその外側のブロックに知られているすべての名前，およびデータ指示転送に有効なすべての名前が含まれている。

   入力の場合に，認識不可能な要素変数またはブロックの中で知られていない名前がストリームに含まれているときは，NAME条件が起こる。ストリームの中に含まれていない名前が，想定されたデータ・リストに含まれている場合は，関連する変数の値はそのまま変らない。

   出力の場合は，想定されたデータ・リストの中のすべての項目が転送される。

3. 入力ストリームの中でセミコロンまたはファイルの終りが認識されると，データ・リストが指定されているかどうかには関係なく，転送は終了する。出力の場合は，各々のPUTステートメントによって最後のデータ項目が転送されたあと，セミコロンがストリームに書き出される。

ストリームの中のデータ指示データ

データ指示転送と結びつけられたストリームの中のデータは，要素割当てのリストの形式になっている。問題データの場合は，次のような一般形式をしている(任意選択の符号をつけた定数も，変数名と等号と同様，文字形式にする)。

要素変数　＝　データ値
〔｛b l ，｝要素変数　＝
データ値〕………；

問題データの一般的な規則は，次の通りである。

1. 要素変数は，添字付の名前であってもよい。添字は，任意選択で符号をつけた１０進数の整定数でなければならない。

2. 入力の場合，要素割当ては，１個のブランク（上記の形式ではb）またはカンマによって区切ることができる。余分のブランクは無視される。出力の場合，割当ては１個のブランクによって区切られる。（PRINT ファイルの場合，各項目は，プログラムのタブ・セット位置にしたがって区切られる。）

3. ストリームの中の各々のデータ値は，リスト指示転送のところで述べた形式のいずれかの形をしている。

4. 入力の場合は，要素割当ての後にセミコロンがあると，１つのGET DATAステートメントを実行することによって転送される要素割当てのリストが終了するので，特定のステートメントによって実際に転送される要素割当ての数は，これによって決まる。出力の場合は，PUT DATAステートメントが完了すると，セミコロンが転送される。

5. ロケーター修飾子が，ストリームの中にあってはならない。基底付変数といっしょに宣言されたロケーター修飾子は，世代を設定するために用いられる。ロケーター修飾子が宣言されなかった基底付変数は，転送することができない。

データ・リストの中にある基底付変数には，次のような制約が適用される。

a. この変数は，オフセット変数を基底とするものであってはならない。

b. この変数は，REFER オプションをつけて宣言された構造のメンバーであってはならない。

c. この変数の基底となっているポインターは，基底付であったり，定義されたものであったり，パラメーターであってはならず，また，配列または構造のメンバーであってはならない。

データ・リストの中にある，定義された変数は，次のようにして定義されたものであってはならない。

a. 制御変数の上に

b. １つまたは２つ以上の調整可能な限界をもつ配列の上に

c. 定数以外のものを指定する，POSITION属性といっしょに，

例

図１１.２に示されている例の中では，Aは，６個の要素をもつ１次元の配列として宣言されており，Bは，７個の要素をもつ１次元の配列である。このプロシージャーは，次の式の計算をして，値を書き出すためのものである。

A（I）＝B（I＋1）≠B（I）；

データ指示のデータ指定（入力の場合）

データ指示入力の一般的な規則

1. データ指定にデータ・リストが含まれていないときは，ストリームの中の名前は，転送の時点で知られているものであれば，どのような名前であってもよい。入力ストリームの中にある修飾された名前は，完全に修飾されたものでなければならない。この名前は，２５６桁を越えてはならない。

2. データ・リストが使用されているときは，データ・リストの各要素は，要素変数，配列変数，または構造変数でなければならない。データ・リストの中で，名前に添字をつけることはできないが，修飾された名前を使用することはできる。ストリームの中にあるすべての名前は，データ・リスト上にもなければならないが，名前の順序は同じである必要はないし，また，ストリームの中にない名前がデータ・リストに含まれていてもよい。たとえば，次のデータ・リスト（A，B，C，Dは要素変数の名前）について考えてみよう。

DATA （B, A, C, D）

このデータ・リストは，次のような入力データ・ストリームと結びつけることができる。

A=2.5， B=.0047， D=125， Z='ABC'；

注： Cは，データ・リストの中にあるが，ストリームの中にはない。したがって，その値は，そのまま変更されない。Zは，データ・リストの中にないので，NAME条件を引き起す。

3. データ・リストに配列の名前が含まれているときは，データ・リストの中に添字付の名前を入れることはできないが，ストリームの中には，その配列に対する添字付の参照があってもよい。配列全体がストリームの中にある必要はなく，ストリームの中に実際にある要素だけが割り当てられる。添字が範囲外にあったり，添字がないときは，NAME 条件が起こる。

```
AB: PROCEDURE;
    DECLARE (A(6),B(7)) FIXED;
    GET FILE (X) DATA (B);
    DO I = 1 TO 6;
    A (I) = B (I+1) + B (I);
    END;
    PUT FILE (Y) DATA (A);
    END AB;
```

入力ストリーム

```
B(1)=1, B(2)=2, B(3)=3,
B(4)=1, B(5)=2, B(6)=3, B(7)=4;
```

出力ストリーム

```
A(1)=3  A(2)=5  A(3)=4  A(4)=3
A(5)=5  A(6)=7;
```

図11.2　データ指示転送の例（入力と出力）

Xは，次のように宣言された２次元の配列の名前であるとしよう。

```
DECLARE X (2,3);
```

次のようなデータ・リストと入力データ・ストリームについて考えてみよう。

データ・リスト	入力データ・ストリーム
DATA (X)	X(1,1)= 7.95, X(1,2)= 8085, X(1,3)= 73;

データ・リストは，配列の名前しかもっていないが，関連する入力ストリームには，配列の個々の要素の値が含まれていてもよい。この場合は，３つの要素だけが割り当てられる。配列の残りの部分は，変らない。

4. データ・リストに構造要素の名前が含まれている場合は，データ・リストの中では完全な修飾は不要であるが，ストリームの中に出てくる名前は，完全に修飾された名前でなければならない。次の構造について考えてみよう。

```
DECLARE 1 CARDIN, 2 PARTNO, 2 DESCRP,
        2 PRICE, 3 RETAIL, 3 WHSL,
        1 CARDOUT, 2 PARTNO, 2 DESCRP,
        2 PRICE, 3 RETAIL, 3 WHSL;
```

CARDIN・PRICE・RETAILの値を読みたい場合は，データ指定と入力データ・ストリームは，次のような形式にすることができる。

データ指定	入力データ・ストリーム
DATA (CARDIN.RETAIL)	CARDIN.PRICE. RETAIL = 4.28;

5. ストリームの中にある修飾された名前の中に，交錯して添字をつけることはできない。添字はすべて，修飾された名前の最後の名前の後にくるように，右側に移さなければならない。たとえば，Yが次のように宣言されているとしよう。

```
DECLARE 1 Y(5,5),2 A(10), 3 B,3 C,3 D;
```

要素名は，ストリームの中で，次のようなものでなければならない。

```
Y.A.B(2,3,8)= 8.72
```

データ・リストの中の名前に，添字を含めることはできない。

データ指示のデータ指定（出力の場合）

データ指示の出力の一般規則は，次の通りである。

1. データ・リストの要素は，要素変数，配列変数，または構造変数のいずれであってもよいし，あるいは，これらの要素のいずれかを含む繰返し指定，またはさらに別の繰返し指定を含む繰返し指定であってもよい。添字付の名前があってもよい。問題データの場合は，データ・リストの中にある名前とその値とが，要素割付けのリストの形式（ブランクで区切られ，セミコロンで終る）で転送される。プログラム制御データの場合は，名前だけが転送される。（PRINTファイルの場合は，各項目は，プログラムのタブ・セット位置にしたがって区切られる。）

2. データ・リストの中にある配列変数は，その中に含まれる添字付の要素が行を主とした順序で並べられたリストとして扱われる。次のように宣言された配列について考えてみよう。

```
DECLARE X (2,4) FIXED;
```

Xがデータ・リストの中で次のようなものであるとすれば，

DATA (X)

出力されると，出力データ・ストリームは次のような形式になる。

X(1,1)= 1 X(1,2)= 2 X(1,3)= 3
X(1,4)= 4 X(2,1)= 5 X(2,2)= 6
X(2,3)= 7 X(2,4)= 8;

注： 実際の出力の中では，2つ以上のブランクが等号の後に出される。コード化算術形式から文字に変換されるとき，先行ゼロはブランクに変換され，最高3個のブランクがフィールドの先頭に出てくることがある。

3. データ・リストの中にある添字式が評価され，その値によって式が置き換えられる。

4. データ・リストの中にある構造の部分である項目は，完全な修飾をつけて転送されるが，添字は，交錯させずに，修飾された名前の後につける。たとえば，データ指示出力のもとで転送される構造の要素に対して，データ・リストが次のように指定されていたとすれば，

DATA (Y(1,-3).Q)

出力ストリームの中の関連するデータ・フィールドは，次のような形式になる。

Y.Q(1,-3)= 3.756;

5. データ・リストの中の構造名は，その中に含まれるいくつかの要素のリストとして解釈され，その中に含まれる配列は，上記のように扱われる。

たとえば，次の構造について考えてみよう。

1 A, 2 B, 2 C, 3 D

データ指示出力のデータ・リストが次のようなものであり，

DATA (A)

BとDの値がそれぞれ2と17であるとすれば，出力ストリームの中の関連するデータ・フィールドは，次のようになる。

A.B= 2 A.C.D= 17;

6. 次のような場合のデータ指示出力は，あとでデータ指示入力に用いても無効である。

a. 数字変数の文字ストリング値が，正しい算術定数（符号は任意選択）を表わしていないとき。たとえば，複素数の数字変数の場合に，このようなことがいえる。

b. プログラム制御データが転送されたとき。

データ指示の出力フィールドの長さ

外部媒体上におけるデータ・フィールドの長さは，その変数に対して宣言された属性と，完全に修飾し，添字をつけた名前の長さ（名前も含まれるのであるから）とによって決まる。コード化算術データ，数字データ，およびビット・ストリング・データから変換される出力項目のフィールドの長さは，リスト指示の出力データの場合と同じであり，文字タイプへのデータ変換の規則によって決められる。この規則については，F節「データの変換と式の値の計算」のところで述べる。

文字ストリング・データの場合は，文字ストリングの内容が，引用符で囲んで書き出される。文字ストリングの中に含まれる引用符はそれぞれ，連続する2つの引用符によって表わされる。

編集指示のデータ指定

編集指示のデータ指定の一般形式は，入力の場合も出力の場合も，次の通りである。

EDIT {(データ・リスト) (形式リスト)}…

1. データ・リストは，上記の「データ・リスト」の項で述べた通りであり，括弧で囲まなければならない。形式リストには，1つまたは2つ以上の形式項目を入れるが，これも括弧で囲まなければならない。形式項目には，次にのべるように3つのタイプがある。ストリーム内のデータを記述するデータ形式項目，ページ，行，およびスペースの操作を記述する制御形式項目，および使用する形式リストを含む別個のステートメントのラベルを指定する遠隔形式項目の3つのタイプである。形式リストと形式項目については，このあとの「形式リスト」の項で詳しく述べる。

2. 入力の場合は，ストリーム内のデータは，連続する文字のストリングと考えられ，個々のデータ項目にわけては考えられない。各データ項目の文字の桁数は，形式リストの中の形式

項目によって指定される。文字は，関連する形式項目にしたがって処理される。

3. 出力の場合，データ・リストの中の各項目の値は，関連する形式項目によって指定された形式に変換され，ストリームの中の1つのフィールド（その幅も，形式項目によって指定されている）に入れられる。

4. 入力の場合も，出力の場合も，最初のデータ形式項目はデータ・リストの中の最初の項目と結びつけられ，2番目のデータ形式項目はデータ・リストの中の2番目の項目と，……というように結びつけられる。形式リストに含まれる形式項目の数が，関連するデータ・リストの中の項目の数より少ないときは，形式リストが再使用されるが，形式項目の方が多いときは，無視される。たとえば，形式リストには5個のデータ形式項目が含まれており，関連するデータ・リストでは10個の項目を転送するよう指定されている場合を考えてみよう。このような場合は，データ・リストの中の6番目の項目が，最初のデータ形式項目と結びつけられる，……という形になる。次に，形式リストに10個のデータ形式項目が含まれており，その関連するデータ・リストが項目を5個しか指定していない場合を考えてみよう。この場合は，6番目から10番目までの形式項目は無視される。

5. データ・リストの中にある配列変数または構造変数は，データ・リストの中の<u>n</u>個の項目に相当する（<u>n</u>は，配列または構造の中にある要素項目の数であり，その各々は，データ形式項目をそれぞれ別個に使用して関連づけられる）。

6. 制御形式項目に出会うと，制御の処置が実行され，データ・リスト項目は，次の形式項目と対にされる。

7. データ・リストの中の最後の項目が，対応する形式項目を用いて処理されたときに，指定された転送が完了する。その後につづく形式項目は，<u>制御形式項目も含めて，</u>無視される。

8. 出力の場合，データ項目は，自動的には区切られないが，通常，算術データ項目には，データ変換の規則とゼロ消去とによって，先行ゼロが含まれる。

<u>例</u>：

```
GET EDIT (NAME, DATA, SALARY)
     (A(N), X(2), A(6), F(6,2));
PUT EDIT ('INVENTORY='||INUM, INVCODE)
     (A, F(5));
```

最初の例で指定されている内容は，ストリームの中の最初のN桁の文字は文字ストリングとして取り扱い，NAME に割り当てること，次の2桁はスキップすること，次の6桁は文字形式でDATA に割り当てること，次の6桁は，符号付（任意）の10進固定小数点定数と考え，SALARYに割り当てること，である。

2番目の例では，'INVENTORY=' という文字ストリングをINUM の値と連結して，ストリームの中の1つのフィールド（その幅はその結果出来上るストリングの長さ）に入れることが指定されている。そのあとで，INVCODE の値が，符号付（任意）の10進固定小数点整定数を表わす文字に変換され，5桁の幅をもつフィールド（先行文字はブランクになることもある）に右づめされて，ストリームに入れられる。式および定数によって表わされる値は，出力データ・リストの中でしか書くことができない点に注意していただきたい。

形式リスト

編集指示データ指定はそれぞれ，形式リストと結びつけなければならない。

一般形式：

（形式リスト）

この"形式リスト"は次のように定義されている。

```
⎧項目          ⎫  ⎡,  項目          ⎤
⎨n項目         ⎬  ⎢,  n項目         ⎥ …
⎩n（形式リスト）⎭  ⎣,  n（形式リスト）⎦
```

統語規則：

1. 各"項目"は，以下にのべるような形式項目を表わす。

2. <u>n</u>という文字は，反復因数を表わしており，括弧で囲んだ式または符号なしの10進整定数のどちらであってもよい。後者の場合は，定数とその後の形式項目をブランクで区切らなければならない。反復因数は，関連する形式項目または形式リストが連続して<u>n</u>回使用されることを指定するものである。ゼロの反復因数は，関連する形式項目または形式リストはスキップされ，使用されないこと（データ・リスト項目は，次の形式項目と結びつけられる）を指定するものである。反復因数を表わすのに式が用いられているときは，1組の反復が

行なわれるたびに，式が評価され，整数に変換される（これは，負の数であってはならない）。関連する形式項目または形式リストは，反復因数のすぐ右側にある項目または項目のリストである。

一般規則：

形式項目には，データ形式項目，制御形式項目，および遠隔形式項目，の3つのタイプがある。データ形式項目は，データ・フィールドがとる外部形式を指定するものである。制御形式項目は，ページ，行，桁，およびスペースの操作を指定するものである。遠隔形式項目は，そのブロック内のどこかに別個の FORMAT ステートメントを設けて，その中で形式項目を指定するためのものである。

各種の形式項目については，E節「編集指示の形式項目」のところで詳しく説明する。以下の説明は，編集指示データ指定での形式項目の用法を示したものである。

データ形式項目

入力の場合，各データ形式項目は，データ項目と結びつけられる文字の桁数と，外部データの解釈の仕方を指定する。データ項目は，必要ならば，変数の属性に合うよう変換されて，データ・リストの中で指名されている関連の変数に割り当てられる。出力の場合は，データ・リストの中の関連する要素の値は，形式項目で指定された文字表現に変換されて，データ・ストリームに挿入される。

データ形式項目には，固定小数点（F），浮動小数点（E），複素数（C），ピクチャー（P），文字ストリング（A），およびビット・ストリング（B）の6つがある。これらは通常，次のように指定される。

F　(w〔，d〔，p〕〕)
E　(w，d〔，s〕)
C　（実数形式項目〔，実数形式項目〕）
P　'ピクチャー指定'
A　〔(w)〕
B　〔(w)〕

このリストの中で，<u>w</u>という文字は，フィールドの桁数を指定する式を表わしている。<u>d</u>という文字は，小数点の右側の桁数を指定するもので，固定小数点整数の場合は省略してもよい。複素数形式項目の実数形式項目は，F，E，またはP形式項目のいずれかの形を表わしている。P形式項目のピクチャー指定は，数字指定または文字ストリング指定のどちらであってもよい。出力の場合，EおよびF形式項目と結びつけられたデータは，内部精度が外部精度より大きければ，まるめられる。

3番目の指定（<u>p</u>）は，F形式項目の中で使用されるもので，スケール因数である。3番目の指定（<u>s</u>）は，浮動小数点数の最初のサブ・フィールドとしてとっておくべき桁数を指定するために，E形式項目の中で使用される。これらの指定については，E節「編集指示の形式項目」のところで詳しく説明する。

<u>注</u>：　固定小数点2進数と浮動小数点2進数のデータ項目はつねに，入力ストリームの中では，10進数で表わされた値によって表現されていなくてはならない。その後，これらをアクセスするために，FおよびE形式項目が用いられ，割当ての際に，その値が2進数表現に変換される。出力の際には，2進数項目が10進数値に変換されるので，関連するFまたはE形式項目では，変換された10進数にもとづいて，フィールドの幅と小数点の位置を記述しなければならない。

次の例は，形式項目の用法を示したものである。

1. GET FILE (INFILE) EDIT (ITEM) (A(20));

   このステートメントは，INFILE と呼ばれるファイルの次の20文字を，ITEM に割り当てる。この値は自動的に，形式項目A(20) によって指定された文字表現から，ITEM に対して宣言された属性によって指定された表現に変換される。

   <u>注</u>：　出力に対してデータ・リストと形式リストが使用されている場合は，フィールドの幅がストリングの長さと同じであるならば，つまり，ストリングの後にブランクをつけないならば，ストリング項目の長さを形式項目の中で指定する必要はない。

2. PUT FILE (MASKFLE) EDIT (MASK) (B);

   MASKがBIT(25)という属性をもっているとすれば，上記のステートメントは，MASK の値を，0と1からなる25桁の文字のストリングとして，MASKFLE という名のファイルに書き出す。フィールドの幅の指定は，B形式項目の中で与えることができる。これは，入力の場合には必ず記述<u>しなければならない。</u>

3. PUT EDIT (TOTAL) (F(6,2));

   TOTAL が FIXED (4,2) という属性をもっているとすれば，上記のステートメントは，TOTAL の値を，固定小数点数の文字表現に変換し，標準出力ファイル SYSPRINTに書くことを指定している。

最後の2つの数字の前に小数点が挿入され，この数は，6桁のフィールドの中に右づめされる。先行ゼロはブランクに変えられ，必要ならば，最初の数字の左に負の符号が入れられる。

内部10進数固定小数点タイプから文字タイプへの変換は，変換に関する普通の規則にしたがって行なわれる。余分な文字があると，変換されたストリングの数字の前にブランクとして出てくる。また，先行ゼロはブランクに変換されるので，追加のブランクが数字の前にあってもよい。小数点または負の符号がある場合は，どちらも1個の先行ブランクと置き換えられる。

編集指示出力では，形式項目の中でフィールドの幅指定を使用して(この場合は，F(6,2) 形式項目の中の6)，先行ゼロを切り捨てることができる。この指定では，ゼロが1つ切り捨てられる。TOTAL は，長さが7の文字ストリングに変換されることになる。変換された数字の4桁がすべてゼロより大きければ，その数は，挿入される小数点も含めて5桁を必要とする。その数が負ならば，負符号のためにもう1桁必要である。したがって，F(6,2)という指定はつねに，すべての数字，小数点，符号を許容することができるが，余分のブランクは切り捨てられて取り除かれる。

4. GET FILE(A) EDIT (ESTIMATE) (E(10,6));

このステートメントは，Aというファイルから次の10文字を取り出し，それらを浮動小数点10進数として解釈するものである。小数点は，小数部の右端6桁の前にあるものとみなされる。データの中に実際の小数点があれば，この仮定を打ち消すことができる。この数の値は，ESTIMATEの属性に合わせて変換され，この変数に割り当てられる。

5. GET EDIT (NAME, TOTAL)
(P'AAAAA',P'9999');

このステートメントが実行されると，標準入力ファイルSYSIN が想定される。最初の5桁は，英字またはブランクでなければならず，これらはNAME に割り当てられる。次の4桁は，ブランク以外の数字でなければならず，これらはTOTAL に割り当てられる。

制御形式項目

制御形式項目には，スペース形式項目(X)，およびCOLUMN, LINE, PAGE, SKIP の各形式項目がある。スペース形式項目は，データ・ストリームの中での相対的なスペース送りを指定するものである。PAGE およびLINE 形式項目は，PRINT ファイルのときにだけ使用可能なものであり，したがって当然，PUT ステートメントの中でしか指定することはできない。通常，PAGE 以外はすべて，式を含んでいる。LINE, PAGE, およびSKIP は，PUT ステートメントの中のオプションとして，別個に指定してもよいし，SKIPは，GETステートメントの中のオプションとして指定することもできる。次にあげる例は，制御形式項目の用法を示したものである。

1. GET EDIT (NUMBER, REBATE)
(A(5), X(5), A(5));

このステートメントは，標準入力ファイルSYSIN からの次の15文字を，以下のようにして処理するものである。最初の5文字は NUMBER に割り当てられ，次の5文字はスペースされて無視され，残りの5文字が REBATE に割り当てられる。

2. GET FILE(IN) EDIT(MAN,OVERTIME)
(SKIP(1), A(6), COLUMN(60), F(4,2));

このステートメントは，INというファイルに結びつけられたデータ・セットを，次の行に位置づけするものである。そして，その行の最初の6文字がMANに割り当てられ，60桁目からの4文字がOVERTIMEに割り当てられる。

3. PUT FILE(OUT) EDIT (PART, COUNT)
(A(4), X(2), F(5));

このステートメントは，PART の値を表わす4文字，次に2桁のブランク，さらに終りに，COUNT の固定小数点数値を表わす5文字を，OUTという名のFILE に入れるものである。

4. 次の例は，COLUMN, LINE, PAGE, およびSKIP 形式項目を他のものと結びつけて使用する方法を示したものである。

```
PUT EDIT ('QUARTERLY STATEMENT')
     (PAGE, LINE(2), A(19));
PUT EDIT
     (ACCT#, BOUGHT, SOLD,
      PAYMENT, BALANCE)
     (SKIP(3), A(6), COLUMN(14),
      F(7,2), COLUMN(30), F(7,2),
      COLUMN(45), F(7,2),
      COLUMN(60), F(7,2));
```

最初のPUTステートメントは，QUARTERLY

STATEMENTという見出しを，標準出力ファイルSYSPRINTの新しいページの2行目に書くことを指定している。2番目のステートメントは，2行スキップ(つまり，"その後の3行目にスキップする")してから，5番目の行の最初の桁からACCT#の値を書くこと，14桁目からBOUGHTの値を書くこと，30桁目からSOLDの値を書くこと，45桁目からPAYMENTの値を書くこと，60桁目からBALANCEの値を書くこと，を指定している。

注：制御形式項目は，形式リストの中に出てきた時点で実行される。データ・リストが使い果されたあとに出てきた制御形式リストは，何も効力をもたない。

遠隔形式項目

遠隔形式項目(R)は，別のところにあるFORMATステートメントのラベル(またはFORMATステートメントのラベルの値をもつラベル式)を指定するものである。FORMATステートメントと，遠隔形式項目を指定するGETまたはPUTステートメントとは，同じブロックの内部になければならない。FORMATステートメントには，遠い場所にある形式項目が含まれている。この機能は，以下の例に示すように，実行時点で別の形式指定を選択することができるようにするものである。

```
DECLARE SWITCH LABEL;
GET FILE(IN) LIST(CODE);
IF CODE = 1
     THEN SWITCH = L1;
     ELSE SWITCH = L2;
GET FILE(IN) EDIT (W,X,Y,Z)
     (R(SWITCH));
L1;   FORMAT (4 F(8,3));
L2;   FORMAT (4 E(12,6));
```

ラベル変数としてSWITCHが宣言されているので，2番目のGETステートメントは，2つのFORMATステートメントのどちらかによって操作することができる。

形式項目の中の式

データ形式項目の中のw，p，d，およびsの指定は，制御形式項目の中の指定と同様，10進整定数である必要はなく，式を用いてもよい。これらは，変数または他の式であってもよい。

変数に読み込まれた値は，のちにデータ・リストの中で別の変数に結びつけて，形式項目の中で使用することができる。

```
PUT EDIT (NAME, NUMBER, CITY)
     (A(N), A(N-4), A(10));
```

```
GET EDIT (M,STRING_A,I,STRING_B)
     (F(2), A(M), X(M), F(2), A(I));
```

最初の例では，NAMEの値が，N桁のフィールドに左づめされた文字ストリングとしてストリームに挿入される。NUMBERは，N－4桁のフィールドの中に左づめされる。CITYは，10桁のフィールドに左づめされる。2番目の例では，最初の2桁がMに割り当てられる。そのあと，STRING_Aに割り当てられる文字の桁数を指定するため，および2桁の文字をIに割り当てる前に無視する桁数を指定するためにも，Mの値がとられる。さらに，STRING_Bに割り当てられる文字の桁数がIの値を用いて，指定されている。

PRINTファイル

PRINT属性は，STREAM OUTPUTファイルにだけ適用することができる。これは，そのファイルのデータが，最終的には印刷されることを(最初は，印刷ページ以外の媒体に書き出してもよい)表わすものである。PRINTファイルの各レコードの最初のデータ・バイトは，印刷制御文字用にとっておかれる。制御オプションと形式項目PAGE，SKIP，およびLINEを含むステートメントが実行されたときに，コンパイラーが自動的に制御文字を挿入する。

PRINTファイルのレイアウトは，図11.3にリストされているオプションと形式項目を用いて制御することができる。(PRINT以外のファイルの場合も，LINESIZE，SKIP，およびCOLUMNを使用することができることに注意していただきたい。)LINESIZEとPAGESIZEは，ページの印刷域の寸法(脚注をのぞく)を決めるものである。LINESIZEオプションは，各印刷行に含まれる最高の文字桁数を指定するものであり，PRINTファイルに対してこれが指定されていないときは，省略時解釈によって120桁の値がとられる(ただし，PRINT以外のファイルの場合には，省略時解釈はない)。PAGESIZEオプションは，各ページに印刷される最高の行数を指定するものであり，指定されていなければ，省略時解釈によって60行とみなされる。次の例について考えてみよう。

```
OPEN FILE(REPORT) OUTPUT STREAM PRINT
  PAGESIZE(55) LINESIZE(110);
```

このステートメントは，REPORTというファイルをPRINTファイルとして開くものである。PAGESIZE(55)という指定は，各ページには，最高55行が含まれなければならないことを表わしている。55行分を書き終った(あるいはスキップした)あとでページ上に書こうとすると，ENDPAGE条件が起こる。ENDPAGE条件に対する標準のシステム処理は，新しいページにスキップすることであるが，プログラマーは，ONステートメントを用いて，自分なりの処置を設定することができる。

オプション	編集指示形式項目	オプションまたは形式項目が記入されるステートメント	効果
LINESIZE(w)[1]	−	OPEN	行の幅を決める
PAGESIZE(w)	−	OPEN	ページの幅を決める
PAGE	PAGE	PUT	新しいページにスキップする
LINE(w)	LINE(w)	PUT	指定の行にスキップする
SKIP〔(x)〕[1]	SKIP〔(x)〕[1]	PUT	指定された行数をスキップする
−	COLUMN(w)[1]	PUT	行の中の指定された桁にスキップする

[1] PRINT 以外のファイルにも使用可能。前記の「転送ステートメントのオプション」と「制御形式項目」，および下記の「行のサイズとレコード形式」の項を参照のこと。

図11.3 PRINTファイルのレイアウトを制御するためのオプションと形式項目

各ページごとに1回だけ，ENDPAGE条件が起こる。したがって，最初にENDPAGE条件が起ったあとも，指定されたPAGESIZEを越えて，印刷をつづけることができる。これは，たとえば，各ページの下の方に脚注を書くような場合に便利である。たとえば，

```
ON ENDPAGE(REPORT) BEGIN;
    PUT FILE(REPORT) SKIP LIST
     (FOOTING);
    N = N + 1;
    PUT FILE(REPORT) PAGE LIST
     ('PAGE'||N);
    PUT FILE(REPORT) SKIP (3);
    END;
```

前の例で示したように，REPORTはPAGESIZE(55)として開かれているものとしよう。56行目に書こうとすると(あるいは55行目以降にスキップしようとすると)，ENDPAGE条件が起こり，上に示されている開始ブロックが実行される。最初のPUTステートメントは，1行分スキップして，FOOTINGの値(おそらくは文字ストリング)を57行目に印刷する(ENDPAGEが起ったときの行は，つねにPAGESIZE+1)ことを指定している。ページ番号が増やされ，REPORT ファイルは次のページにセットされ，文字ストリング'PAGE' が新しいページ番号と連結されて，印刷される。最後のPUTステートメントは，3行分のスキップをするので，次の行は4行目になる。開始ブロックから，ENDPAGE条件を引き起したPUTステートメントに制御が戻り，データが印刷される。しかし，そのステートメントの中で指定されたSKIP オプションはもはや効力をもたない。

ENDPAGEオン・ユニットがないときは，SIGNAL ENDPAGEは無視されることに注意していただきたい。

LINESIZE(110) という指定は，ページ上の各行に最高110桁を収容できることを示すものである。110桁を越える行を書こうとすると，余分な文字は，次の行に入れられることになる。

標準ファイルSYSPRINT

標準ファイルSYSPRINTは，明示的に宣言されていない限り，つねにPRINT属性が与えられる。ファイルが開かれるときに，自動的に新しいページが開始される。そのファイルを参照する最初のPUTステートメントがPAGEオプションをもっている場合，または最初のPUTステートメントに，最初の項目としてPAGEをもつ形式リストが含まれている場合は，ブランクのページができることになる。

ENVIRONMENT属性

ENVIRONMENT属性は，ファイルと結びつけられるデータ・セットの物理的な構成についての情報を指定するものである。この情報は，括弧に入れたオプション・リストの中に入れる。その一般形式は，次の通りである。

ENVIRONMENT （オプション・リスト）

ストリーム方式の転送で使用できるオプションは，以下の通りである。

MEDIUM （記号装置名〔，物理装置タイプ〕）
〔FUNCTION(a〔，b〕)〕
〔RCE〕
〔STACKER(n)〕
〔F|FB|V|VB|D|DB|U〕
〔RECSIZE(レコードの長さ)〕
〔BLKSIZE(ブロック・サイズ)〕
〔BUFFERS({1|2})〕
〔CONSECUTIVE〕
〔LEAVE〕
〔UNLOAD〕
〔NOTAPEMK〕
〔NOLABEL〕
〔VERIFY〕
〔ASCII〕
〔BUFOFF〔(n)〕〕

これらのオプションは，どのような順序で記入してもよく，ブランクで区切って記入する。オプション自体の中にブランクを含めてはならない。

次に，これらのオプションについて説明する。

装置のタイプ（MEDIUM）

PL/Iのファイル宣言の各々に対して，MEDIUMオプションを指定しなければならない。このオプションについては，第10章「入力と出力」のところに説明してある。

装置の機能（FUNCTION）

FUNCTIONオプションは，3525で実行される機能を指定するものである。このオプションについては，第10章「入力と出力」のところに説明してある。

読取桁抑制（RCE）

ENVIRONMENT属性のRCEオプションは，3505または3525によるカード桁の選択的な読取りを指定するものである。このオプションを使用するときは，そのプログラムによって読み取られるカードのデックの最初のカードは，RCE形式記述カードでなければならない。形式記述カードの形式については，このコンパイラー用のプログラマー手引書をみていただきたい。

1. MEDIUMオプションの中で指定されている装置タイプは，3505，3525，または3525Tでなければならない。
2. MEDIUMオプションの中で指定されている論理装置は，SYSIPTであってはならない。
3. 光学式マーク読取りオプション(OMR)を指定してはならない。

スタッカー選択（STACKER）

STACKER属性は，3505または3525のどのスタッカーにカードを落すかを指定するものである。このオプションは，次のような形式をしている。

STACKER(n)

nは，定数か，または精度(31，0)のFIXED BINARY STATIC変数である。

1. MEDIUMオプションの中で指定されている装置タイプは，3505，3525，または3525Tでなければならない。
2. FUNCTIONオプションで，書出し(FUNCTION(w))を指定してはならない。
3. nは，1または2でなければならない。

STACERオプションが指定されていないときは，STACKER(1)とみなされる。

レコード形式オプション

ストリーム方式の転送では，レコードの境界は無視されるが，データ・セットを作成するときは，レコード形式は重要な問題である。その理由としては，レコード形式如何によって，データ・セットが占める記憶域スペースの量や，そのデータを処理するプログラムの効率が左右されるばかりでなく，このデータ・セットはのちに，レコード方式の転送によって処理されることもあることなどがあげられる。レコード形式を指定したあと，プログラマーは，ストリーム方式の転送だけを使用している限りは，レコードやブロックについて考慮する必要はない。つまり，プログラマーは，いくつかの行

の中に並べられた一連の文字としてそのデータ・セットを考えればよく，新しい行を選択するためには，SKIP オプションまたは形式項目（PRINT ファイルの場合は，さらにPAGE および LINE オプションと形式項目）を使用することができる。

レコードは，次の7つの形式のいずれかにすることができる。

固定長：	F	非ブロック化
	FB	ブロック化
可変長：	V	非ブロック化
	VB	ブロック化
	D	非ブロック化（「ASCII データ・セット」の項を参照）
	DB	ブロック化（「ASCII データ・セット）の項を参照）
不定長：	U	ブロック化することはできない。

レコードのブロック化および非ブロック化は，自動的に行なわれる。

どのような形式のものであっても，レコードはすべて，データ・バイトと，任意選択の制御バイトまたは接頭語バイトから構成されている。可変長レコードには，レコードまたはブロックの長さを指定するための制御バイトと接頭語バイトが含まれる。これらのバイトの用法については，このあとで説明する。さらに，PRINT ファイルに結びつけられるレコード（どのような形式であっても）の先頭には，印刷制御文字として解釈されるデータ・バイトがついている。コンパイラーは，関連するPUTステートメントを分析し，適当な文字（または省略時解釈の文字）を挿入する。

固定長レコード

データ・セットの中のレコードはすべて，同じ長さである。

F一形式： レコードはブロック化されていない。1つのレコードがそれぞれ1個のブロックを構成する。

FB一形式： レコードはブロック化されている。いくつかのブロックは短ブロックであってもよい。つまり，指定されたブロック・サイズより短くてもよい。

可変長レコード

各レコードは，長さが異なっていてもよい。

V一形式： レコードはブロック化されていない。1つのレコードがそれぞれ1個のブロックを構成する。各レコードは，次のものから構成されている。

4バイトの制御バイト

データ・バイト

4バイトの制御バイトには，レコードの長さ（つまり，現在のレコードの長さ）が入っている。この値は自動的に挿入されるものであり，プログラマーは何の処置もとる必要はない。

さらに，ブロック（つまり，レコード）の先頭に特別の4バイトがつけられる。これらのバイトには，ブロック・サイズが入っている。この値は，レコードの長さの場合と同じようにして挿入される。

VB一形式： レコードはブロック化されている。各レコードは次のもので構成されている。

4バイトの制御バイト

データ・バイト

4バイトの制御バイトの使用目的は，V一形式レコードの場合と同じである。

ブロックは，V一形式レコードの場合と同様，ブロック・サイズを入れるための4バイトの特別な制御バイトをもっている。

D一および DB一形式： 「ASCII データ・セット」を参照のこと。

不定長レコード

処理はすべて，プログラマーの責任である。長さの指定がレコードの中に必要な場合は，プログラマーがそれを準備し，解釈もしなければならない。

RECSIZEオプション

RECSIZEオプションは，レコードの長さを指定するものである。これは，次にあげるものの合計になる。

1. データに必要な長さ。可変長と不定長レコードの場合は，最大の長さにする。

2. 必要とされる制御バイト。可変長レコードの場合は，レコー

ドの長さ用に4バイト必要。固定長および不定長レコードでは不要。

レコードの長さは，10進整定数として，またはFIXED BINARY(31,0) STATICという属性をもつ変数として，指定することができる。

この値は，次の約束を守ったものでなければならない。

最大： 固定長および不定長の場合： 32,760バイト

可変長の場合： 32,756バイト ASCIIデータ・セットの場合はつねに，最高9999バイトである。

ゼロの値： 省略時解釈が適用される（「レコード形式の省略時解釈」の項を参照）。

負の値： UNDEFINEDFILE条件が起こる。

LINESIZEオプションによって暗示される値は，RECSIZE オプションの中で指定された値を置き換える。

BLKSIZEオプション

BLKSIZEオプションは，ブロックサイズを指定するものである。これは，次のものの合計になる。

1. ブロック内の全レコードの長さ。可変長レコードの場合は，各レコードの長さには，レコードの長さ用の4バイトの制御バイトも含まれる。

2. 必要とされる制御バイトまたはブロック接頭語バイト（ASCII データ・セットのとき）。可変長ブロック化レコードでは，ブロック・サイズ用に4バイトが必要である。固定長および不定長レコードでは，不要である。

ブロックサイズは，10進整定数として，あるいはFIXED BINARY(31,0) STATICという属性をもつ変数として指定することができる。

この値は，次の約束を守ったものでなければならない。

最大： 32,760バイト（BUFOFF(L)が指定されているASCIIデータ・セットの場合は，9999）。

ゼロの値： 省略時解釈が適用される（このあとの「レコード形式の省略時解釈」を参照のこと）。

負の値： UNDEFINED条件が起こる。

ブロック・サイズとレコードの長さとの関係は，レコード形式によって決まる。

FB−形式 ブロック・サイズは，レコードの長さの倍数でなければならない。

VB−形式 ブロック・サイズは，次のものの合計と同じかそれ以上でなければならない。

ブロック内の全レコードの長さ

ブロック・サイズ用の4バイトの制御バイト

DB−形式 ブロック・サイズは，次のものの合計と同じかそれ以上でなければならない。

ブロック内の全レコードの長さ

ブロック接頭語バイト（あれば）の長さ

レコード形式の省略時解釈

ENVIRONMENT属性の中でレコード形式オプションが何も指定されていないときは，次の処置がとられる。

INPUT ファイル： MEDIUMオプションの中で，記号装置名SYSIPTまたはユニット・レコード装置が指定されているときの省略時解釈は，レコード形式がF，レコードの長さ，ブロック・サイズ，LINESIZEのいずれもが80になる。それ以外のINPUTファイルの場合はすべて，次のようになる。

レコード形式： エラー・メッセージが作成される。

ブロック・サイズまたはレコードの長さ： どちらか一方が指定されているときは，他方の値は，指定されたオプションの値から推論される（制御接頭語バイトが必要であったり，ついていたりする場合は，それらを加えたり，引いたりした上で）。どちらも指定されていないときは，UNDEFINED条件が起こる。

OUTPUTファイル： MEDIUMオプションの中で記号装置名SYSPCH またはカード穿孔装置が指定されているときの省略時解釈は，レコード形式がF，LINESIZEが80，レコードの長

さとブロック・サイズの両方が８１になる。記号装置名ＳＹＳＬＳＴ，または行印刷装置が指定されているときの省略時解釈は，レコード形式がＦ，ＬＩＮＥＳＩＺＥが１２０，レコードの長さとブロック・サイズの両方が１２１になる。これ以外の OUTPUT ファイルの場合は，次のようになる。

レコード形式：　ＶＢ－形式にセットされる（ ASCII オプションが指定されているときはＤＢ－形式）。

レコードの長さ：

PRINT ファイル：

Ｆ，ＦＢ，またはＵ：　ＬＩＮＥＳＩＺＥ+1

Ｖ，ＶＢ，ＤまたはＤＢ：　ＬＩＮＥＳＩＺＥ+5

この場合は，ＬＩＮＥＳＩＺＥとして指定された値または省略時解釈された値が使用される。

PRINT 以外のファイル：

Ｆ，ＦＢ，またはＵ：　ＬＩＮＥＳＩＺＥ

Ｖ，ＶＢ，Ｄ，またはＤＢ：　ＬＩＮＥＳＩＺＥ+4

この場合は，ＬＩＮＥＳＩＺＥとして指定された値が用いられる。省略時解釈の値はない。

ブロック・サイズ：　ＦＢ：　５＊（レコードの長さ）
ＶＢ：　５＊（レコードの長さ）＋4

緩衝域オフセット：　Ｆ，ＦＢ，またはＵ：　0
Ｄ，またはＤＢ：　4
ＤＢ：　５＊（レコードの長さ）＋（ブロック接頭語バイト）

注：

1. PRINT ファイルの場合は，ＬＩＮＥＳＩＺＥの省略時解釈の値は１２０である。PRINT 以外のファイルの場合は，ＬＩＮＥＳＩＺＥの省略時解釈はない。

2. 上記のようにして計算した値が 32,７６０ より大きい場合は，ブロック・サイズは，レコードの長さ＋４にセットされ，レコード形式はＶにセットされる。ASCII データ・セットの場合に，省略時解釈のブロック・サイズが 32,７００，または9999（ＢＵＦＯＦＦが接頭語の長さの値をもたずに指定されているとき）より大きいと，ブロック・サイズは（レコードの長さ＋ブロック接頭語）にセットされ，レコード形式はＤにセットされる。

3. 出力ファイル上のＤＢ－形式レコードの場合は，ブロック接頭語の長さ（つまり，緩衝域オフセット）はつねに，０または４のどちらかでなければならない。

緩衝域の割振り

緩衝域は，データ・セットとの間で転送されるデータの中間的な記憶域として用いられる内部記憶域である。緩衝域を使用すると，転送の時間と計算の時間とをオーバーラップさせることができる。レコードのブロッキングおよびデブロッキングを自動的に行なう場合に，緩衝域は重要な役割をする。

BUFFERSオプション

ＢＵＦＦＥＲＳオプションは，データ・セットに対して割り振られる緩衝域の数を指定するものである。この数は，１または２でなければならない。ストリーム・ファイルの場合は，省略時解釈として２つの緩衝域が用いられるが，ユーザーが，この数を１に変えることもできる。

データ・セットの編成

データがどのような形でデータ・セットに記録されるか，また，その後，プログラムに転送できるようにするためにデータがどのようにして取り出されるか，ということは，データ・セットの編成方法の如何によって決まる。このコンパイラーは，データ・セットの編成として，CONSECUTIVE，INDEXED，およびREGIONALの３つを認識することができる。ストリーム方式の転送によってアクセスされるデータ・セットは，CONSECUTIVE 編成のものでなければならないが，これは，データ・セット編成の省略時解釈であるから，STREAM ファイルに対してとくに指定する必要はない。

CONSECUTIVEデータ・セット

CONSECUTIVEデータ・セットの中のレコードは，書き出された順序通りに並べられており，これらは，それと同じ順序でしか取り出すことはできない。データ・セットが作成されたあと，関連するファイルを，入力のとき（データを読むために），または出力のときに（末尾にレコードを追加することによってデータ・セットを拡張するため，またはデータ・セットの内容を新しいデータで置き換えるために），開くことができる。

磁気テープの取扱いオプション

LEAVEオプション

このオプションは，後で逆方向の読取りをするファイルの場合に用いられるものであり，ファイルが閉じられるときに，テープ・ボリュームの巻戻しをさせないようにする働きをもつ。

UNLOADオプション

UNLOADオプションは，ファイルが閉じられるときに，テープを巻き戻し，アンロードすることを指定するために用いられる。

NOTAPEMKオプション

テープ・ファイルに対してNOTAPEMKオプションを指定すると，プログラマーは，ラベルのないテープ・ファイル上のデータ・レコードの前に先行テープ・マークを書かせないようにすることができる。

NOLABELが指定されているテープOUTPUT ファイルに対して，NOTAPEMKを使用することができる。UNBUFFEREDファイルの場合は，このオプションは許されない。

NOLABELオプション

NOLABELオプションは，磁気テープ・ファイルに対して，ファイル・ラベルを処理しないことを指定するために用いられる。

出力ファイルに対してNOLABEL オプションが指定されている場合は，ENVIRONMENT属性の中で，NOLABELのほかに，NOTAPEMKが指定されていない限り，テープ・マークが自動的に書き出される。標準外ラベルおよび追加のユーザー・ラベルは処理されない。

ASCIIデータ・セット

ASCII を使用する磁気テープのデータ・セットを，PL/I で作成したり，アクセスしたりすることができる。このコンパイラーでは，F，U，およびDレコード形式がサポートされている。FおよびU形式は，他のデータ・セットの場合と同じように扱われる。DおよびDB形式は，他のデータ・セットの場合のVおよびVB形式に相当するもので，これについて以下に説明する。

レコード形式のほかに，2つのENVIRONMENTオプション（ASCIIおよび緩衝域オフセット・オプションBUFOFF）を指定することができる。

ASCIIオプション

このオプションは，データ・セット上のデータを表わすのに用いられるコードがASCII であることを指定するものである。

BUFOFFオプションおよびブロック接頭語フィールド

ASCII データ・セット内の各ブロックの先頭には，ブロック接頭語フィールドと呼ばれるフィールドを設けることができる。これは，1バイトから99バイトまでの長さにすることができる。緩衝域オフセット・オプションは，このフィールドの長さをデータ管理プログラムに表示して，各物理ブロックの先頭からのこのオフセットから，データのアクセスまたは作成が開始されるようにするためのものである。PL/Iは，このフィールドへのアクセスをサポートしておらず，また，通常，システム/360およびシステム/370のコンパイラーで使用される情報も含まれていない。このブロック接頭語に含まれる情報をデータ管理プログラムが使用する状況は1つだけで，非ブロック化またはブロック化可変長レコード（つまり，D－またはDB－形式レコード）の場合に，ブロックの長さを記録するためにブロック接頭語フィールドを使用することができる。この場合のフィールドは，4バイトの長さで，ブロック接頭語フィールド自体も含めてブロックの長さをバイト単位で示した，右づめの10進数文字値を含んでいる。このあと，これは，ブロックの長さフィールドと全く同じになる。

緩衝域オフセット・オプションの形式は，BUFOFF〔(n)〕である。接頭語の長さに相当する数値は，"n"として指定する。これは，10進整定数，またはFIXED BIMARY(31,0)，STATIC という属性をもつ変数のどちらかとして，指定することができる。その最小値はゼロで，最大値は99である。接頭語の長さの指定がないことは，ブロック接頭語がブロックの長さフィールドとして用いられることを示しており，この場合は，フィールドの長さが4バイトであることが暗示される。ブロックの長さは，データ管理プログラムによって，接頭語に挿入される。

入力の場合，1から99バイトの長さのブロック接頭語フィールドをもっていても，ブロック接頭語フィールドを全くもっていなくても，ASCIIデータ・セットをアクセスすることができる。また，ブロックの長さフィールドとしてブロック接頭語フィールドが使用されているかどうかには関係なく，ASCIIデータ・セットをアクセスすることができる。出力の場合は，3つの有効なレコード形式のいずれか1つを使用するデータ・セットは，ブロック接頭語をつけずに作成することができるが，ブロック接頭語の作成がPL/Iによってサポートされる唯一の状況は，それがブロックの長さフィールドとして使用される場合である。

BUFOFFオプションは，ASCIIデータ・セットの場合にだけ使用することができる。

D－形式およびDB－形式レコード

各レコードの長さは，異なっていてもよい。この2つの形式は，次の通りである。

D—形式：　レコードはブロック化されていない。1つのレコードが1つのブロックを構成する。各レコードには，次のものが含まれる。

4バイトの制御バイト

データ・バイト

4バイトの制御バイトには，レコードの長さが含まれている。この値は，データ管理プログラムが挿入するので，プログラマーは何の処置もとる必要はない。さらに，ブロックの先頭にブロック接頭語フィールドがつけられることもあるが，ここには，ブロックの長さが入る場合もある。

DB—形式：　レコードはブロック化されている。D形式に対して与えられる，これ以外の情報はすべて，DB形式にも与えられる。

省略時解釈の規則

「レコード形式の省略時解釈」のところで述べた規則のほかに，さらに次のような規則が適用される。

ENVIRONMENTオプションの中でASCIIは指定されていないが，BUFOFF，D，またはDBのいずれか1つが指定されているという場合は，ASCIIとみなされる。

直接アクセス装置の用法

VERIFYオプション

VERIFYオプションは，書出し操作が行なわれるたびにそのあとで読取りチェックを行なうことを指定するために，用いられるものである。このオプションは，直接アクセス装置の場合にだけ，使用することができる。

第12章： レコード方式の転送

この章では，レコード方式の転送で使用される入出力ステートメントについて解説する。レコード方式とストリーム方式の転送のどちらにも等しく適用されるPL/Iの機能（ファイル，ファイル属性，ファイルの開閉など）については，第10章「入力と出力」のところですでに説明した。

レコード方式の転送では，データ・セットの中のデータは，オペレーティング・システムに受け入れられる形式で記録されたレコードの集まりとみなされる。レコード方式の転送の間に，データ変換は行なわれない。入力のときに，READ ステートメントが実行されると，1つのレコードが，データ・セットの中で記録されている通りの形でプログラム変数に転送されるか，または緩衝域の中にある該当のレコードを指すようポインターがセットされる。出力のときは，WRITE，REWRITE，またはLOCATEステートメントが実行されると，1つのレコードが，内部的に記録されている通りの形でプログラム変数から転送される。データは，実際には，ブロック単位でデータ・セットとの間で転送されるが，レコード方式の転送で用いられるステートメントが関係するのは，レコードだけであり，レコードは自動的にブロック化またはデブロック化される。

転送されるデータ

パラメーターや DEFINED 変数も含めて，殆んどの変数は，レコード方式の転送ステートメントによって転送することができるが，一般に，この章で述べることは，すべての変数に対して等しく適用されるものである。2，3のタイプのデータについては，特別な考慮事項があり，これらについて以下に説明する。

データの集合体

データの集合体に対して，次のような制約が適用される。

1. 集合体は，連結された記憶域に入っていなければならない（集合体パラメーターは，CONNECTED属性をもっていなければならない）。

2. LOCATEステートメントの場合，変数は，レベル1の基底付変数でなければならない。

境界合せされないビット・ストリング

次にあげるものは，転送することができない。

1. 境界合せされていない固定長ビット・ストリングである，BASED，DEFINED，パラメーター，添字付，または構造 - 基底 - 要素の各変数。

2. 最初または最後の基底要素が，境界合せされていない固定長ビット・ストリングであるような小構造（ただし，これらが，外側の大構造の最初または最後の要素でもある場合はのぞく）。

3. BASEDまたはDEFINED属性をもっているか，またはパラメーターであるような大構造，およびそれらの最初または最後の要素として，境界合せされていない固定長ビット・ストリングをもつ大構造。

長さの変わるストリングとエリア変数

長さの変わるストリングを指定する位置指定モードのステートメントは，ストリングの最大の長さと，ストリングの現在の長さを示す2バイトの接頭語とを加えたものに等しい長さをもつフィールドを転送する。このファイルに対しては，SCALARVARYINGオプションを指定しなければならない。エリア変数を指定する位置指定モードのステートメントは，エリアに対して宣言されたサイズと，制御情報の入っている16バイトの接頭語とを加えたものに等しい長さをもつフィールドを転送する。

長さの変わるストリングを指定する移動モードの出力ステートメントは，ストリングの現在の長さだけを転送する。2バイトの接頭語は，SCALARVARYING オプションがファイルに対して指定されているときにだけ，含まれる。要素エリア変数，または最後の要素がエリア変数であるような構造を指定する移動モードのステートメントは，エリアの現在のエクステントに16バイトの接頭語を加えたものだけを転送する。

データ転送ステートメント

次に，レコード方式のデータ転送ステートメントの概要を説明する。これらについての詳細は，J節「ステートメント」のところで解説されている。

実際に，補助記憶装置との間でレコードを転送するステートメン

トは4つある。これらは，READ, WRITE, LOCATE およびREWRITEである。5番目のステートメントであるDELETEステートメントは，UPDATEファイルからのレコードを削除するために用いられるものである。どのステートメントが使用可能であるかは，ファイルの属性によって決まる。

READステートメント

READ ステートメントは，どのようなINPUTまたはUPDATEファイルに対しても使用することができる。これは，データ・セットからプログラムへ(直接的に変数へ，あるいは緩衝域へ)レコードを転送する働きをする。ブロック化レコードの場合は，READステートメントは，緩衝域から変数にレコードを転送するか，または緩衝域の中の該当するレコードを指すようポインターをセットする。したがって，どのREAD ステートメントも，入力装置からデータを転送するというわけではない。

WRITEステートメント

WRITE ステートメントは，OUTPUTファイルまたはDIRECT UPDATEファイルに対して使用することができるが，SEQUENTIAL UPDATEファイルに対しては使用することができない。このステートメントは，プログラムからデータ・セットへレコードを転送する働きをする。非ブロック化レコードの場合は，変数から直接転送することもできるし，緩衝域から転送することもできる。ブロック化レコードの場合は，WRITE ステートメントは，論理レコードを緩衝域に入れ，さらに，レコードのブロッキングが完了したときにだけ，出力装置に対して実際にデータ転送を行なう。

REWRITEステートメント

REWRITEステートメントは，UPDATE ファイル内のレコードを置き換える働きをする。SEQUENTIAL UPDATE ファイルの場合は，REWRITE ステートメントは，そのファイルから読みとられた最後のレコードを書き直すことを指定する。したがって，レコードは，書き直しをする前に，読んでおかなければならない。DIRECT UPDATE ファイルの場合は，レコードは，先に読まれているかどうかには関係なく，書き直すことができる。

LOCATEステートメント

LOCATEステートメントは，OUTPUT SEQUENTIAL BUFFERED ファイルの場合にだけ使用することができる。このステートメントは，出力緩衝域の中で，基底付変数のための記憶域を割り振り，緩衝域内に割り振った位置にポインターをセットする。このあと，このポインターを使用して割振りを参照することにより，データを緩衝域に移動することができる。論理レコードのブロック全体が緩衝域の中にある場合は，ブロックは出力装置に転送される。

DELETEステートメント

DELETEステートメントは，UPDATE ファイル内のレコードを削除することを指定するものである。

転送ステートメントのオプション

レコード方式のデータ転送ステートメントで使用できるオプションは，ファイルの属性および結びつけられるデータ・セットの性質によって異なる。各々のタイプのファイルで使用できる組合せのすべてのリストが，この章の後の方にある図12.5，12.6，および12.7に示してある。

各々のオプションは，キーワードの後に値をつけたものからなっている。この値は，ファイル式，変数，または式を括弧で囲んだものである。どのステートメントの場合も，オプションはどのような順序で記入してもよい。

FILEオプション

FILE オプションは，すべてのレコード方式ステートメントの中で記入しなければならない。このオプションは，その操作が行なわれるファイルを指定するものである。これは，キーワードFILEの後に，括弧で囲んだファイル式をつづけて記入するという形になっている。このセクションの個々のステートメントの説明のところに，FILE オプションの例が示してある。

INTOオプション

INPUTまたはUPDATEファイルに対するREAD ステートメントの中で，INTO オプションを使用することができる。INTOオプションは，論理レコードが読み込まれる変数を指定するものである。

```
READ FILE (DETAIL) INTO (RECORD_1);
```

これは，次の順番のレコードをRECORD_1という変数に読み込むことを指定している。

INTO オプションは，長さの変わる要素ストリング変数を指名することができる点に注意していただきたい。そのファイルに対して，ENVIRONMENT属性のSCALARVARYING オプションが指定されているときは，ストリング・データの長さを指定

する2バイトの接頭語が各レコードに含まれているものとみなされる。(以下の「FROM オプション」の項を参照。)

SCALARVARYING オプションが指定されなかったときは，入力の場合，コンパイラーは，レコードの長さからストリングの長さを計算し，それを2バイトの接頭語としてつける。長さの変わるストリングの場合は，この計算の結果，長さは8の倍数に切り上げられるので，計算された長さが，宣言された最大の長さよりも大きくなることもある。

FROMオプション

OUTPUTまたはDIRECT UPDATEファイルに対するWRITE ステートメントの中では，FROM オプションを使用しなければならない。また，UPDATE ファイルに対するREWRITEステートメントの中でも使用することができる。FROM オプションは，レコードがそこから書き出される変数を指定するものである。

FROMオプションは，長さの変わる要素ストリング変数を指名することができる点に注意していただきたい。FROMオプションをもつWRITE ステートメントを使用するときは，長さの変わるストリングの現在の長さだけがデータ・セットに転送されるが，長さを指定する2バイトの接頭語がつけられることもある。この接頭語は，そのファイルに対してENVIRONMENT属性のSCALARVARYING オプションが指定されているときにだけ，つけられる。

SEQUENTIAL BUFFERED UPDATE ファイルの場合は，REWRITEステートメントのFROMオプションを省略することができる。最後のレコードが，INTO オプションをもつREAD ステートメントによって読まれた場合は，FROM をもたないREWRITEは，データ・セット内のレコードに対して何の影響ももたないが，最後のレコードが，SETオプションをもつREAD ステートメントによって読まれた場合は，行なわれた何らかの割当てによって，レコードは緩衝域の中で更新され，データ・セットに書き戻される。

```
WRITE FILE (MASTER) FROM (MAS_REC);
REWRITE FILE (MASTER) FROM (MAS_REC);
```

上のステートメントは両方とも，変数MAS_RECの値がファイルMASTERに書き出されることを指定している。WRITE ステーメントの場合は，SEQUENTIAL OUTPUT ファイルの中の新しいレコードを指定している。REWRITE ステートメントは，SEQUENTIAL UPDATE ファイルから読まれた最後のレコードを，MAS_REC によって置き換えることを指定している。

SETオプション

READ ステートメントまたはLOCATEステートメントで，SETオプションを使用することができる。これは，READ 操作中にデータが入れられた緩衝域の中の位置，またはLOCATE ステートメントによって割り振られた緩衝域内の位置を指すよう，指名されたポインター変数をセットするよう指定するものである。

```
READ FILE(X) SET (P);
```

このステートメントは，緩衝域の中にある次の順番のレコードの位置に，ポインター変数Pの値をセットするよう指定するものである。LOCATEステートメントの中でSETオプションが省略されている場合は，そのレコード変数といっしょに宣言されているポインターがセットされる。

長さの変わる要素ストリング変数を転送する場合は，そのファイルに対してSCALARVARYINGオプションを指定しなければならないことに注意していただきたい。

IGNOREオプション

SEQUENTIAL INPUT またはSEQUENTIAL UPDATE ファイルに対するREAD ステートメントの中で，IGNORE オプションを使用することができる。このオプションは，スキップして無視すべきレコードの数を指定する整数値をもつ式を指定するものである。式の値が負またはゼロのときは，レコードはスキップされない。

```
READ FILE (IN) IGNORE (3);
```

このステートメントは，ファイルの中の次の3つのレコードをスキップするよう指定している。

READ ステートメントに，INTO，SET，およびIGNORE オプションのいずれもが含まれていないときは，IGNORE(1)とみなされる。

KEYオプション

KEYオプションは，INDEXED，REGIONAL，またはVSAM編成のデータ・セットと結びつけられたKEYEDファイルに対してだけ適用される。(レコード方式の転送で使用できるデータ・セット編成のタイプについては，この章の後の方の「データ・セット編成」の項で説明する。)このオプションは，キーワードKEYの後に括弧に入れた式(文字ストリング定数，変数またはそれ以外の要素式のいずれでもよい)をつづけた形にする。必要なら，式が評価され，文字ストリングに変換される。文字ストリングの長さおよびそれが表わすものに関する規則については，この章の後の方の「INDEXED編成」および「REGIONAL編成」のところで説明する。

KEYオプションは，特定のレコードを識別するものである。このオプションは，INPUTまたはUPDATEファイルに対するREADステートメントの中，またはDIRECT UPDATEファイルに対するREWRITEステートメントの中で使用することができる。(結びつけられたデータ・セットの編成が，INDEXEDであるか，またはキーの順に並べられたVSAM編成の場合だけは，SEQUENTIALファイルに対するREADステートメントの中で，KEYオプションを使用することができる。)

```
READ FILE (STOCK) INTO (ITEM)
  KEY (STKEY);
```

このステートメントは，変数STKEYの文字ストリング値によって識別されるレコードを，変数ITEMに読み込むことを指定するものである。

KEYFROMおよびKEYTOオプション

KEYFROMおよびKEYTOオプションは，INDEXED，REGIONAL，またはVSAM編成のデータ・セットと結びつけられたKEYEDファイルにだけ適用されるものである。各々のオプションは，KEYFROMまたはKEYTOというキーワードの後に括弧に入れた式をつづける形になっている。KEYFROMの場合の式は，文字ストリング定数，またはその他の要素式にすることができる。必要ならば，式は評価され，文字ストリングに変換される。KEYTOの場合の式は，長さが255バイト以下の文字ストリング変数でなければならない。文字ストリングの長さおよびそれらが表わしているものに関する規則については，以下の「INDEXED編成」および「REGIONAL編成」の項で述べる。

KEYFROMオプションは，データ・セット上の該当レコードを識別するキーを指定するものである。このオプションは，REGIONAL編成のSEQUENTIAL OUTPUTファイル，DIRECT UPDATEファイル，DIRECT OUTPUTファイルに対するWRITEステートメント，またはLOCATEステートメントの中で使用することができる。

```
WRITE FILE (LOANS) FROM (LOANREC)
  KEYFROM (LOANNO);
```

このステートメントは，LOANRECの値を1つのレコードとしてファイルLOANSの中に書き出すこと，LOANNOの文字ストリング値を，後でそのレコードを取り出すためのキーとして使用することを指定している。

KEYTOオプションは，レコードのキーを読み込む変数の値を指定するものである。これは，SEQUENTIAL INPUTまたはSEQUENTIAL UPDATEファイルに対するREADステートメントの中で使用することができる。

```
READ FILE (DETAIL) INTO (INVTRY)
  KEYTO (KEYFLD);
```

このステートメントは，ファイルDETAILの中の次のレコードを変数INVTRYに読み込むこと，およびそのレコードのキーは変数KEYFLDに読み込むことを指定している。

EVENTオプション

EVENTオプションは，キーワードEVENTの後に括弧に入れた事象変数の名前をつづけて書く形になっている。(EVENTオプションの中に名前を記入すると，それは，事象変数を文脈的に宣言したことになる。)CONSECUTIVE，REGIONAL，またはVSAM編成のUNBUFFEREDファイル，またはすべてのDIRECTファイル(ファイルがVSAMのキー順の編成のときはUNBUFFERED)に対するREAD，WRITE，またはREWRITEステートメントの中に，このオプションを記入することができる。VSAMキー順の編成のUNBUFFEREDファイルに対するDELETEステートメントの中にも，このオプションを記入することができる。

EVENTオプションは，入出力操作を非同期的に行なうこと(つまり，その間に他の処理がつづけられること)，および同じ事象変数を指定するWAITステートメントが実行されるまで，入出力条件(UNDEFINEDFLLEをのぞく)を起さないことを指定するものである。たとえば，

```
READ FILE (MASTER) INTO (REC_VAR)
     EVENT (RECORD_1);
          ⋮
WAIT (RECORD_1);
```

READステートメントのオプションの中の式が評価され終ると，入力操作が開始される。このあとすぐに，後のステートメントが実行される。制御がWAITステートメントに達するまで，RECORD，TRANSMIT，KEY，またはENDFILE条件は起らない。WAITステートメントが実行されたときに入力操作が完了していない場合，および4つの条件が1つも起っていない場合は，操作が完了するまで，その先のステートメントの実行は延期される。操作が正しく完了すると，WAITステートメントのすぐ後にあるステートメントから処理が続行される。READステートメントを実行した結果，4つの条件のいずれかが起っている場合は，WAITステートメントが実行されたときに，その条件がひき起される。このコンパイラーを使った場合は，TRANSMITとRECORD条件だけが，同時に起り得る。TRANSMITは，つねに最初に処理される。そのあと，実行されたオン・ユニットから正常に戻ると，WAITステートメントのすぐ後のステートメントから，処理が続行される。EVENTオプションは，非同期処理を指定するものではあるが，4つの条件はいずれも，WAITステートメントによっ

て処理と同期されるまで，割込みを起すことはできない。

1つのファイルに対する未解決の入出力操作は，1つしか許されないことに注意していただきたい。あるファイルに対する前の入出力操作がまだWAIT されていないうちに，同じファイルに対して入出力操作を開始しようとすると，ERROR 条件が起こる。

EVENT オプションをもつステートメントが一旦実行された後は，そのオプションの中で指名された事象変数は，活動状態にあるとみなされる。これが活動状態にある間は，同じ事象変数を再びEVENT オプションの中で指定することはできない。事象変数は，対応するWAIT ステートメントが実行されたあと，またはファイルが閉じられたときに，非活動状態に戻る。

DISPLAYステートメントでも，EVENTオプションを使用することができる。

完了値と状況値

事象変数は，2つの別々の値をもっている。事象が完了しているかどうかを示す完了値，および事象が正常に完了したかどうかを示す状況値の2つである。完了値は，1つのビットであるが，状況値は，省略時解釈の精度（15，0）をもつ固定2進数である。非同期式の入出力操作が実行されると，完了値は'0'B（「未完了」を表わす）にセットされ，状況値はゼロ（「異常なく完了」を表わす）にセットされる。非同期式の入出力操作が完了し，関連するWAIT ステートメントが実行されると，完了値は'1'Bにセットされ，さらに，非同期式入出力操作を実行した結果，入出力条件が引き起されたときは事象変数の状況値は1（「異常終了」）にセットされる（まだそのときゼロであったならば）。完了値は，COMPLETION組込み関数によってとり出すことができるし，さらに非活動状態の事象変数の場合は，COMPLETION擬似変数によってセットすることができる。状況値は，STATUS組込み関数によって得ることができるし，STATUS 擬似変数によってセットすることができる。

処理モード

レコード方式の転送では，プログラマーに対して，データの取扱いに関して2つの方法が提供されている。プログラマーは，宣言された記憶域の中でデータを処理することができるが，この方法では，データが実際に主記憶域の中へ（または外へ），直接的にあるいは緩衝域を通って移動されるので，移動モードと呼ばれる。もう1つの方法では，プログラマーは，データが緩衝域の中にある間に処理することができる（つまり，宣言された記憶域にデータを移動しない）。この場合は，データ転送ステートメントを実行すると，緩衝域の中でレコードに割り振られた記憶域の位置が識別されるだけなので，位置指定モードと呼ばれる。位置指定モードは，BUFFEREDファイルの場合にだけ使用可能である。どちらのモードが用いられるかは，プログラマーが使用しているデータ転送ステートメントとオプションによって決まる。

移動モード

移動モードでは，READ ステートメントが与えられると，レコードは，外部記憶装置から，INTO オプションで指名されている変数に転送される（BUFFEREDファイルが使用されているときは，入力緩衝域を通って）。WRITEまたはREWRITE ステートメントが与えられると，レコードは，FROM オプションで指名されている変数から外部記憶装置に転送される（多分，出力緩衝域を通って）。INTO オプションおよびFROM オプションで指名する変数は，どのような記憶域クラスのものであってもよい。

次の例（図12.1に示されている）について，考えてみよう。

```
NEXT:  RFAD FILE(IN) INTO(DATA);
         :
       WRITE FILE(OUT) FROM (DATA);
       GO TO NEXT;
```

最初にREAD ステートメントが実行されたときは，ファイルINに結びつけられたデータ・セットから入力緩衝域に1つのブロックが転送され，そのブロックの最初のレコードが変数DATA に移動される。さらにその後でREAD ステートメントが実行されると，連続するレコードが緩衝域からDATA に移動される。緩衝域の中にあったすべてのレコードの処理が終ったときは，次のREAD ステートメントが与えられると，新しいブロックがデータ・セットから転送されるが，大抵の場合，実際には代替の緩衝域の中にある新しいレコードがアクセスされ，データ転送と処理のオーバーラップができるようになっている。WRITE ステートメントの場合も，出力緩衝域の中に物理レコードが作成され，緩衝域がいっぱいになるたびにファイルOUT と結びつけられたデータ・セットに転送される，というようにして同様の方法で実行される。

移動モードは，緩衝域の境界合せの問題がないので，位置指定モードよりも使い方が簡単である。さらにまた，同じレコードの内容を何回も参照するという場合は，位置指定モードで使用される間接アドレス指定の技法のときに生じるようなオーバーヘッドがないので，実行時間も速くなることもある。

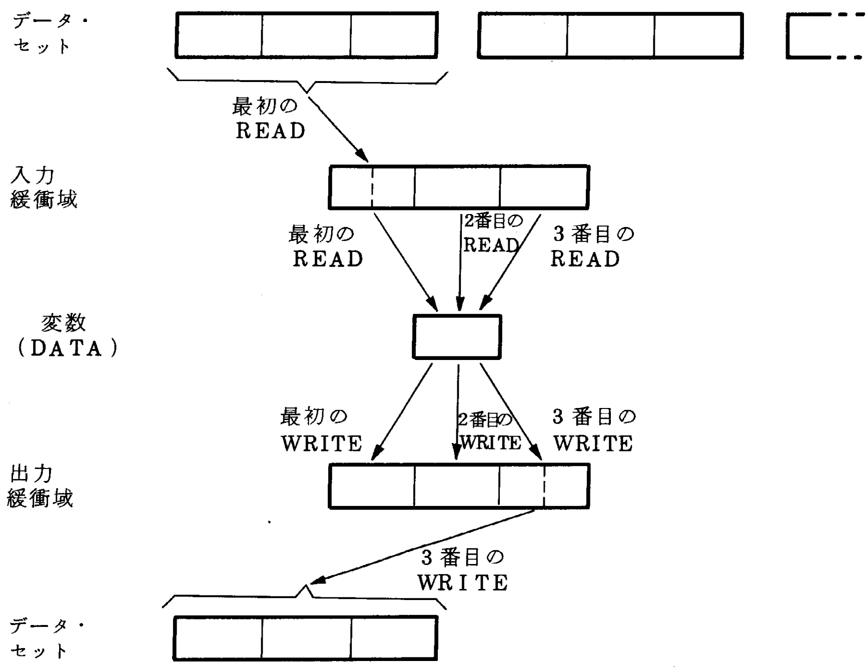


図12.1　入力と出力：移動モード

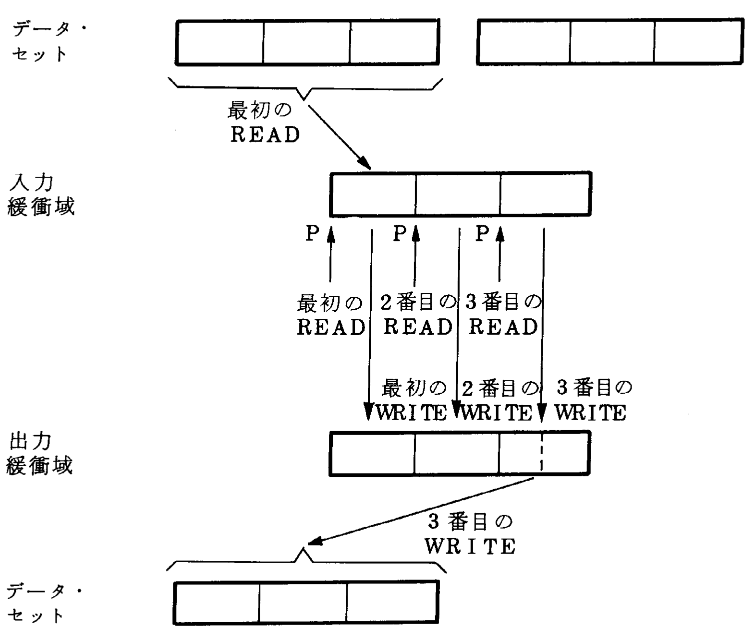


図12.2　位置指定モードの入力，移動モードの出力

位置指定モード

位置指定モードでは，基底付変数を使用する必要がある。緩衝域の中のデータに基底付変数を効果的にオーバーレーすることができるので，それぞれに同じポインターを結びつけることによって，異なる基底付変数を用いて同じデータをアクセスすることができる。したがって，同じデータを別の方法で解釈することができるわけである。位置指定モードは，自己定義のレコードを読むためにも使用することができる。この場合は，レコードの一部分にある情報を用いて，レコードの残りの部分の構造が表示される。たとえば，この情報は，サブフィールドの操返しの回数のカウントであってもよいし，レコードを解釈するためにどのクラスの構造を使用すればよいかを示すコードであってもよい。

READ ステートメントが与えられると，データ・セットから入力緩衝域に１ブロックのデータが転送され（必要なら），さらに，緩衝域にある次のレコードの位置を指すよう，SETオプションで指名されているポインター変数がセットされる。このあと，レコード内のデータは，このポインター変数と結びつけられている基底付変数を参照することによって，処理することができる。このレコードは，同じファイルを参照する次のREAD ステートメントが実行されるまでの間だけ，使用可能である。

LOCATE ステートメントが与えられると，基底付変数に対する記憶域が出力緩衝域の中に割り振られ，割り振られた記憶域を識別するポインターがセットされる。その結果，基底付変数は，それに割り当てられた値をもつようになる。同じファイルに対して次のLOCATE，WRITE，またはCLOSEステートメントが与えられると，必要なら 出力緩衝域の中のデータがデータ・セットに転送される。

位置指定モードでは，データの移動が少なく，また記憶域も少なくてすむので，移動モードの場合よりも実行時間が速くなることが多い。しかし，このモードは充分注意して使用しなければならない。とくに，プログラマーは，緩衝域の中でデータがどのように境界合せされているか，構造のデータがどのようなマップになっているかということに注意しなければならない。構造のマップおよびデータの境界合せについては，K節「データマップの作成」のところで説明する。

図１２.２は，SEQUENTIALファイルに対する以下の例を図示したものであるが，ここでは，入力に対しては位置指定モード，出力に対しては移動モードが使用されている。

```
          DCL DATA BASED(P);
              :
    NEXT: READ FILE(IN) SET(P);
              :
          WRITE FILE(OUT) FROM(DATA);
          GO TO NEXT;
```

最初にREAD ステートメントが実行されると，ファイルINに結びつけられたデータ・セットから入力緩衝域に１つのブロックが転送され，緩衝域の中の最初のレコードを指すよう，ポインター変数Pがセットされる。このあと，変数DATA またはポインターPによって修飾されたその他の基底付変数を参照すると，実際に，この最初のレコードを参照することになる。さらにこのあと，READ ステートメントを実行すると，緩衝域の中の後続のレコードを指すようポインター変数Pがセットされる。緩衝域の中にあるすべてのレコードの処理が終ると，次のREAD ステートメントは，データ・セットから，新しいブロックを転送する。

上の例の中で，入力と出力の両方に位置指定モードを使用したとしても，効率が上がるかどうか疑わしい。入力に移動モードを使用し，出力に位置指定モードを使用する方法もある。たとえば，

```
          DCL DATA BASED(P);
              :
    NEXT: LOCAT DATA FILE(OUT);
          READ FILE(IN) INTO(DATA);
              :
          GO TO NEXT;
```

LOCATEステートメントが実行されるたびに，基底付変数DATAを新しく割り振るために出力緩衝域の中に記憶域が確保され，この記憶域を指すようポインター変数Pがセットされる。最初にREAD ステートメントが実行されると，ファイルINに結びつけられたデータ・セットから入力緩衝域にブロックが転送され，ブロックの中の最初のレコードがDATAの最初の割振りに割り当てられる。そのあと READステートメントが実行されると，後続の論理レコードが，そのときのDATAの割振りに割り当てられる。緩衝域の中にあるすべてのレコードの処理が終ると，次のREADステートメントによって新しいブロックがデータ・セットから転送される。各レコードは，READステートメントを実行してから次にLOCATEステートメントが実行されるまでの間，使用することができる。出力緩衝域の中に使用可能なスペースがなくなったときは，次のLOCATEステートメントが実行されると，ファイルOUTに結びつけられたデータ・セットにブロックが転送され，新しい緩衝域が割り振られる。

READ ステートメントがENDFILE 条件になったときは，ファイルが閉じられるときに転送される緩衝域がファイルOUTに割り振られているはずであることに注意していただきたい。

ENVIRONMENT属性

ENVIRONMENT属性は，ファイルに結びつけられるデータ・セットの物理的な編成に関する情報を指定するものである。この情報は，括弧に入れたオプション・リストに入れる。一般形式は次の通りである。

ENVIRONMENT （オプション・リスト）

レコード方式の転送に適用されるオプションは，以下の通りである。

```
MEDIUM （記号装置名〔，物理装置タイプ〕）
〔FUNCTION(a〔，b〕)〕
〔ASSOCIATE(ファイル名)〕
〔RCE〕
〔OMR〕
〔COLBIN〕
〔STACKER(n)〕
〔F|FB|V|VB|D|DB|U〕
〔RECSIZE(レコードの長さ)〕
〔BLKSIZE(ブロック・サイズ)〕
〔BUFFERS({1|2})〕
 CONSECUTIVE
 INDEXED
 REGIONAL({1|3})
 VSAM
〔PASSWORD(パスワード指定)〕
〔LEAVE〕
〔UNLOAD〕
〔NOTAPEMK〕
〔NOLABEL〕
〔VERIFY〕
〔EXTENTNUMBER〔(n)〕〕
〔INDEXMULTIPLE〕
〔HIGHINDEX(2311|2314|2321|3330|3340)〕
〔OFLTRACKS(n)〕
〔GENKEY〕
〔CTLASA〕
〔CTL360〕
〔COBOL〕
〔INDEXAREA(索引区域サイズ)〕
〔NOWRITE〕
〔ADDBUFF(n)〕
〔SCALARVARYING〕
〔KEYLENGTH(n)〕
〔KEYLOC(n)〕
〔ASCII〕
〔BUFOFF〔(n)〕〕
```

10進整定数の引数を必要とするENVIRONMENT属性のオプションの殆んどのものに関して（ブロック・サイズやレコードの長さ，など），定数または変数を使用することができる。EXTENTNUMBERおよびBUFFERSオプションの場合は，定数が必要である。この場合の変数は，FIXED BINARY (31，0)とSTATIC 属性をもち，添字や修飾をつけていないものでなければならない。

これらのオプションは，どのような順で記入してもよいが，ブランクで区切る。オプション自体はブランクを含んでいてはならない。

これらのオプションについて，以下に説明する。

装置タイプ（MEDIUM）

VSAM ファイルの場合をのぞいて，PL/Iの各ファイルの宣言では，MEDIUM オプションを指定しなければならない。このオプションについては，第10章「入力と出力」のところで許しく説明してある。

装置の機能（FUNCTION）

FUNCTIONオプションは，3525で実行される機能を指定するものである。このオプションについては，第10章「入力と出力」のところで説明してある。

ASSOCIATEオプション

ENVIRONMENT属性のASSOCIATEオプションは，IBM 3525で複数操作を行なう（たとえば，読取りののちに穿孔）ことができるようにするものである。必要な操作ごとに1つずつファイルを宣言し，ASSOCIATEオプションを使って，いくつかのファイルを互いに「結びつける」。

ASSOCIATEオプションは，次のような形式をしている。

ASSOCIATE（ファイル名）

1. "ファイル名"は，同じ外部プロシージャーの中で宣言されたファイル定数でなければならない。

2. MEDIUM オプションの中で指定する装置タイプは，3525か3525Tでなければならない。

3. MEDIUM オプションの中で指定する論理装置は，SYSLST，SYSIPT，またはSYSPCHであってはならない。

4. ファイルは，RECORD属性をもっていなければならない。

5. カードの読取りまたは穿孔に用いられるファイルに対して，明示的に，または省略時解釈により，BUFFERS(1)を指定しなければならない。

6. 結びつけられるいくつかのファイルのセットに対応する2番目のオペラントといっしょに，FUNCTIONオプションを使用しなければならない(第10章「入力と出力」を参照)。

ファイルは，「読取一穿孔一印刷一読取」の順に結びつけなければならない。次の例は，カードを読み取り，穿孔し，さらにその上に印刷するための3つのファイルに対する宣言を示したものである。

```
DCL READ FILE RECORD INPUT
  ENV(…FUNCTION(R,RPW), ASSOCIATE(PUNCH)…),
PUNCH FILE RECORD OUTPUT
  ENV(…FUNCTION(P,RPW), ASSOCIATE(PRINT)…),
PRINT FILE RECORD OUTPUT
  ENV(…FUNCTION(W,RPW), ASSOCIATE(READ)…);
```

結びつけられたファイルのセットに対する入出力操作は，ファイルが結びつけられた順に行なわなければならない。上の例にあげた3つのファイルの場合，正しい順序は次の通りである。

```
READ FILE (READ) ………………;
WRITE FILE (PUNCH)……………;
WRITE FILE (PRINT)……………;
```

プログラマーは，正しい順序が守られているかどうか確認しなければならない。ただし，この順序に印刷操作が含まれている場合はゼロ(印刷操作を省略)から，その装置で許される最大の数(2から25)まで何回でも，印刷操作を連続して行なうことができる。

入／出力の順序の最初のファイルは，暗示的に開いてもよい。残りのファイルは，結びつけられたファイルのいずれかに対して操作が行なわれる前に明示的に開かなければならない。

結びつけられたファイルのセットに対して行なわれるすべての入出力操作が完了する前に，結びつけられたファイルのいずれかが閉じられることのないよう，プログラマーは確認しなければならない。このような理由から，位置指定モードの入出力を使用しているときは，結びつけられたファイルのセットを閉じるために，複数閉止のステートメントを使用するとよい。

読取桁抑制(RCE)

ENVIRONMENT属性のRCEオプションは，3505または3525による。カード桁の選択的な読取りを指定するものである。このオプションを使用する場合は，そのプログラムによって読み取られるカード・デックの最初のカードは，RCE形式記述カードでなければならない。形式記述カードの形式は，このコンパイラーに対するプログラマーの手引書で説明されている。

1. MEDIUM オプションで指定する装置タイプは，3505，3525，または3525Tでなければならない。

2. MEDIUM オプションで指定する論理装置は，SYSIPTであってはならない。

3. 光学式マーク読取操作(OMR)を指定してはならない。

光学式マーク読取(OMR)

ENVIRONMENT属性の OMRオプションは，3505によって，標準の80桁カードの中のマークを光学的に読み取ることを指定するものである。このオプションを使用しているときは，そのプログラムによって読み取られるカード・デックの最初のカードは，OMR形式記述カードでなければならない。形式記述カードの形式は，このコンパイラーのプログラマーの手引書で説明されている。

1. ファイルは，RECORD属性をもっていなければならない。

2. 明示的，または省略時解釈によって，BUFFERS(1)を指定しなければならない。

3. MEDIUM オプションの中の装置タイプは，3505でなければならない。

4. 読取桁抑制オプション(RCE)を指定してはならない。

80(EBCDICのとき)または160(カラム・バイナリーのとき)より小さいブロックサイズ(BLKSIZE)が指定されているときは，ブロックサイズは，80または160のどちらか適当な方に変更される。

カラム・バイナリー(COLBIN)

ENVIRONMENT属性のCOLBINオプションは，3505または3525によって処理されるカードのデータは，カラム・バイナリー形式で収容されていることを指定するものである。

1. ファイルは，RECORD属性をもっていなければならない。

2. MEDIUM オプションの中の装置タイプは，３５０５，３５２５，または３５２５Ｔでなければならない。

3. FUNCTIONオプションで，穿孔翻訳（FUNCTION（P，I））または書出し（FUNCTION（w））を指定してはならない。

スタッカー選択（STACKER）

ENVIRONMENT属性のSTACKERオプションは，３５０５または３５２５のどのスタッカーにカードを落すかを指定するものである。このオプションは，次のような形式をしている。

STACKER（n）

ここで，nは，定数，または精度（31，0）のFIXED BINARY STATIC変数である。

1. MEDIUM オプションで指定する装置タイプは，３５０５，３５２５，または３５２５Ｔでなければならない。

2. FUNCTIONオプションで書出し（FUNCTION(w)）を指定してはならない。

3. nは，１または２のどちらかでなければならない。

STACKER オプションが指定されていないときは，STACKER（１）とみなされる。

レコード形式オプション

レコードの形式は，次にあげる７つの形式のいずれかにすることができる。

固定長：	F	非ブロック化
	FB	ブロック化
可変長：	V	非ブロック化
	VB	ブロック化
	D	非ブロック化 （「ASCII データ・セット」を参照）
	DB	ブロック化 （「ASCII データ・セット」を参照）
不定長：	U	ブロック化することはできない。

レコードのブロッキングおよびデブロッキングは，自動的に行なわれる。

どの形式のものであっても，レコードはすべて，データ・バイトと，任意選択の制御バイトから構成されている。可変長レコードには，レコードとブロックの長さを指定するための制御バイトが含まれている。これらの制御バイトの用法については，このあとで説明する。さらに，どのレコードも（形式の如何を問わず）任意選択により，最初のデータ・バイトに印刷または機械制御文字を入れておくことができる。この文字は，プログラマーが自分で挿入しなければならず，また，ENVIRONMENT属性のCTLASA またはCTL360 オプションを用いて，このような文字があることを表示しなければならない。

どの形式のレコードの場合も，SCALARVARYING オプション（このあと，説明する）を指定することができる。最初のバイトに印刷または機械制御文字が含まれているときは，このオプションを指定することはできない。指定すると，このバイトの解釈が不明確になる。

固定長レコード

データ・セットの中のレコードはすべて，同じ長さである。

F－形式：　レコードはブロック化されていない。１つのレコードが各々１ブロックを構成している。

FB－形式：　レコードはブロック化されている。ブロックのあるものは，短ブロック（指定されたブロック・サイズより短い）であってもよい。

可変長レコード

各レコードの長さは，異なっていてよい。

V－形式：　レコードはブロック化されていない。１つのレコードがそれぞれ，１ブロックを構成している。各レコードは次のものからなっている。

４バイトの制御バイト

データ・バイト

４バイトの制御バイトには，レコードの長さ（４バイトの制御バイトも含めて現在のレコードの長さ）が入っている。この値は自動的に挿入されるので，プログラマーは，何も処置をとる必要はない。さらに，４バイトの余分の制御バイトが，ブロックの先頭に入れられる。これらのバイトには，すべての制御バイトも含めたブロックサイズが含まれる。この

値は，同じようにしてレコードの長さに挿入される。

VB—形式　レコードはブロック化されている。各レコードは，次のものから構成される。

4バイトの制御バイト

データ・バイト

4バイトの制御バイトは，V—形式レコードの場合と同じ目的をもっている。さらに，ブロックは，V—形式レコードの場合と同様，ブロック・サイズ用として4バイトの余分な制御バイトをもっている。

D—および DB—形式：「ASCII データ・セット」の項を参照。

不定長レコード

処理はすべて，プログラマーの責任において行なわれる。レコードの中に長さを指定する必要があれば，プログラマーが準備し，解釈しなければならない。

RECSIZEオプション

RECSIZEオプションは，レコードの長さを指定するものである。これは，次のものの合計になる。

1. データに必要な長さ。可変長および不定長レコードの場合は，最大の長さにする。
2. 必要な制御バイト。可変長レコードの場合は，レコードの長さ用に4バイト必要。固定長および不定長レコードの場合は不要。

レコードの長さは，10進整定数，またはFIXED BINARY (31,0) STATIC属性をもつ変数として指定することができる。

この値は，次の規則に従う。

最大：　固定長および不定長：32,760バイト
可変長：32,756バイト
ASCII データ・セットでは，最大9999

ゼロの値：　省略時解釈の処置がとられる（「レコード形式の省略時解釈」の項を参照）。

負の値：　UNDEFINEDFILE 条件が起こる。

BLKSIZEオプション

BLKSIZEオプションは，データ・セット上の最大のブロックサイズを指定するものである。ブロックの長さは，次のものの合計になる。

1. 次のいずれかの合計の長さ

   1個のレコード
数個のレコード

   可変長レコードの場合は，レコードの長さ用の制御バイト（4バイト）を各レコードの長さに含める。

2. 上のほかに必要な制御バイト，またはブロック接頭語バイト（ASCII データ・セットの場合）。可変長ブロック化レコードの場合は，ブロック・サイズ用に4バイト必要。固定長および不定長レコードでは不要。

   BLKSIZEの値は，10進整定数，またはFIXED BINARY(31,0) STATIC属性をもつ変数として指定することができる。

   この値は，次の規則に従う。

   最大：　32,760バイト（接頭語の長さの値をもたないBUFOFFが指定されているASCII データ・セットの場合は，9999）

   ゼロの値：　省略時解釈の処置がとられる（このあとの「レコード形式の省略時解釈」の項を参照）。

   負の値：　UNDEFINEDFILE 条件が起こる。

ブロック・サイズとレコードの長さの関係は，レコード形式によって異なる。

FB—形式　ブロック・サイズは，レコードの長さの倍数でなければならない。

VB—形式　ブロック・サイズは，次の合計と同じかそれ以上でなければならない。

レコードの最大の長さ

4バイトの制御バイトの長さ

DB一形式	ブロックサイズは，次のものの合計と同じかそれより大きいものでなければならない。 レコードの最大の長さ ブロック接頭語（ついている場合）の長さ

注： VSAMデータ・セット（この章の後の方で説明する）に結びつけられるファイルの場合に，使用できる唯一のレコード形式オプションは，RECSIZE だけである（データ・セットの実際のレコード・サイズと一致していなければならない）。そのほかのもの（つまり，BLKSIZE オプション，およびブロッキングのタイプを示す各種の文字の組合せ）は，指定しても無視される。

レコード形式の省略時解釈

レコード形式のオプションが何も指定されていない場合は，次の処置がとられる。

レコード形式：	エラー・メッセージが出される。
ブロックサイズまたはレコード形式：	どちらか一方が指定されているときは，指定されたオプションから，他方の値が推測される（制御バイトまたは接頭語バイトがあれば，それらが加えられたり，引かれたりする）。どちらも指定されていないときは，UNDEFINEDFILE 条件が起こる。

緩衝域の割振り

緩衝域は，データ・セットとの間で転送されるデータの中間的な記憶域として使用される主記憶域である。緩衝域を使用すると，SEQUENTIALファイルの処理の速度を上げることができる。緩衝域は，レコードの自動的なブロッキングおよびデブロッキングや，位置指定モードの転送にとって，重要な意味をもっている。

BUFFERSオプション

BUFFERSオプションは，データ・セットに割り振られる緩衝域の数を指定するものである。この数は，1または2でなければならない。順次アクセスをするCONSECUTIVE または INDEXED RECORD ファイルの場合は，省略時解釈により2つの緩衝域が与えられるが，ユーザーがこれを1に変えることもできる。これ以外のレコード・ファイルの場合はすべて，省略時解釈は1であり，ユーザーがこれを変えることはできない。

データ・セット編成

データ・セットの編成方式の如何によって，データ・セット・ボリュームにデータがどのように記録されるか，また，後でデータをプログラムに転送するためにどのようにして取り出すか，ということが決まる。レコードは，連続する物理的な位置または論理的な位置をもとにして順次的に，またはデータ転送ステートメントの中で指定されているキーを用いて直接的に，データ・セットに貯えられたり，そこから取り出されたりする。このような貯えかた，取り出しかたとして，PL/Iには，4つの一般的なデータ・セット編成方法がある。この4つは，CONSECUTIVE, INDEXED, REGIONAL，およびVSAMで，データ・セット編成が指定されていないときは，省略時解釈によってCONSECUTIVEとみなされる。

CONSECUTIVE編成のデータ・セットの場合は，レコードは，それらの連続する物理的な位置だけをもとにして編成されており，順次的な順序でだけ取り出される（キーは使用されない）。INDEXEDデータ・セットのレコードは，各々のレコードに結びつけられたキーにしたがって論理的な順に並べられている。レコードは，キーの昇順に並んでおり，データ・セットの中には索引が継続されていて，レコードの取出しに用いられる。REGIONAL編成のデータ・セットは，いくつかの区域に分割されている。この区域の各々は，区域番号によって識別され，それぞれ1個以上のレコードが入れられている。取出しの際には，指定されたキーによって区域番号が与えられ，そこからレコードの探索が始められる。

CONSECUTIVEデータ・セットは，3つのタイプの中では，作成や使用の方法が最も簡単なものであり，外部記憶装置の必要量が少ないという利点をもっている。しかし，CONSECUTIVE データ・セットのレコードは，現在並んでいる順序でしか更新することができず，レコードを挿入したいときは，新しいデータ・セットを作らなければならない。磁気テープの場合，更新の方法はサポートされていない。

INDEXEDデータ・セットは，順次的に作成しなければならないが，一旦出来上ったあとは，ランダムな取出し，更新，追加が可能である。INDEXED データ・セットを順次的に処理した場合は，その中のレコードが物理的な順に取り出されるとは限らないので，これに相当するCONSECUTIVEデータ・セットの場合よりも，速度が遅くなる。さらにまた，ランダムにアクセスする場合も，INDEXED データ・セットでは，レコードを探し当てるために索引を探索しなければならないので，REGIONALデータ・セットにくらべて効率が悪い。このほかに，INDEXED データ・セットの短所として，CONSECUTIVEデータ・セットより

も補助記憶装置のスペースが余計必要であることや，順次処理をするときでも，複数ボリューム・データ・セットの場合はすべてのボリュームをのせておかなければならないこと，があげられる。

REGIONALデータ・セットは，直接的にアクセスしたときは，INDEXED データ・セットの場合よりも速いが，順次的に処理すると，レコードはランダムな順に提示されるという短所をもっている。順次的な取出しの順序は，必ずしもレコードが提示される順序とは一致しないし，また相対的なキーの値と関係をもっている必要もない。REGIONALデータ・セットでは，レコードをブロック化することはできない。

注： キーワードVSAMは，そのファイルがVSAMデータ・セット（この章の後の方で説明する）に結びつけられることを表示するために使用される。VSAMデータ・セットは，キー順またはエントリー順のどちらかの編成になっている。通常，これらの編成は，PL/IのINDEXEDおよびCONSECUTIVE編成にそれぞれ対応する。

PASSWORDオプション

PASSWORDオプションは，VSAMデータ・セットと結びつけられるファイルに対してだけ使用される。VSAMデータ・セットがシステムに対して定義されるときに（アクセス方式サービスのDEFINE 指令を用いて），READおよびUPDATEパスワードをこれに結びつけることができる。そのあとで，このデータ・セットをアクセスするために使用されるPL/Iファイルの宣言の中に，該当するパスワードを含めなければならない。このオプションの形式は，以下の通りである。

PASSWORD（パスワードの指定）

ここで，パスワードの指定は，文字ストリング定数またはSTATIC属性をもつ文字ストリング変数である。この文字ストリングは，8桁になるように補われたり，切り捨てられたりした上で，検査のためVSAMに渡される。正しいパスワードを指定するためのチャンスが何回か，システム操作員に対して与えられる。許されるチャンスの回数は，データ・セットを定義するときに指定する。この回数まで試行がうまくゆかないと，UNDEFINEDFILE条件が起こる。

入出力操作の最適化

一般に，入出力操作は，コンパイルされたコードから呼び出されたライブラリー・サブルーチンによって，行なわれる。しかし，ある特定の条件のもとでは，コンパイラーは，このような操作を実行するためのインライン・コードを作り，ライブラリー呼出しのオーバーヘッドを節減することもできる。このような場合は，入出力ステートメントの実行速度は相当速くなる。

インラインで実行される入出力ステートメントの場合，アクセスまたは作成されるデータ・セットはCONSECUTIVEでなければならず，使用されるファイルは，SEQUENTIAL，BUFFERED，の両属性と，INPUTまたはOUTPUTのどちらかの属性をもつ，非パラメーターのファイル定数でなければならない。ENVIRONMENT属性では，SCALARVARYINGオプション（可変長ストリングが転送される場合），およびF－，FB－，またはU－レコード形式のいずれかのオプションを指定しなければならない。したがって，ファイルの宣言は以下のようになる。

```
DCL F FILE RECORD SEQUENTIAL BUFFERED
       INPUT|OUTPUT ENV(CONSECUTIVE
       F|FB|U 〔SCALARVARYING〕
       〔RECSIZE(xx)〕〔BLKSIZE(xx)〕);
```

標準の省略時解釈の属性とオプションには，下線が引いてある。下線のある属性のうち最低1つは指定しなければならない。指定しないと，省略時解釈によって，ファイルにはSTREAM属性が与えられる。

ファイルを印刷するときは，印刷－穿孔制御オプションCTLASAまたはCTL360のどちらかも，ENVIRONMENT属性で指定しなければならない。

READ SETステートメントは，U－形式レコードを転送するためにBACKWARDS属性が使用されている場合をのぞいて，つねに，インライン・コードによって実施される。それ以外のレコード入出力ステートメント，すなわちREAD INTO，WRITE FROM，およびLOCATEの場合は，次の条件に合っているときは，インライン・コードが生成される。

1. レコード変数の宣言に，ストリングの長さ，配列の限界，またはエリア・サイズが含まれている。

2. F－またはFB－形式レコードのときは，レコード・サイズ，またはU－形式レコードのときはブロックサイズが，ENVIRONMENT属性で，指定されている。

入出力ステートメントを実施するために，インライン・コードが使用されている場合は，コンパイラーが情報メッセージを出す。

INDEXEDデータ・セットをアクセスする際に，INDEXAREA，NOWRITE，およびADDBUFFオプションを用いることによって，入出力操作の速度を上げることができる。詳細については，この章の後の方の「データ管理の最適化」のところで説明する。

磁気テープの取扱いオプション

LEAVEオプション

LEAVE オプションは，ファイルを開くときまたは閉じるときに，巻戻し操作を行なわないことを指定するために用いられる。このオプションは，BACKWARDS属性をもつファイルに対して与えられるものであり，それ以外の場合は，テープは巻き戻される。

UNLOADオプション

UNLOADオプションは，ファイルを閉じるときにテープを巻き戻して，アンロードすることを指定するために用いられる。

NOTAPEMKオプション

テープ・ファイルに対して NOTAPEMK オプションを指定すると，プログラマーは，ラベルなしのテープ・ファイル上のデータ・レコードの前に先行テープ・マークが書かれないようにすることができる。その結果，そのデータ・セットは，ASCII データ・セットの場合をのぞいて，逆方向の読取りが不可能になる。

NOLABEL が指定されているテープ OUTPUT ファイルに対して，NOTAPEMK を使用することができる。UNBUFFEREDファイルに対してこのオプションを使用することはできない。

NOLABELオプション

NOLABEL オプションは，磁気テープ・ファイルに対して，ファイル・ラベルの処理をしないことを指定するために用いられる。

出力ファイルに対して NOLABEL オプションが指定されていると，ENVIRONMENT属性の中で，NOLABEL のほかに NOTAPEMKオプションが指定されていない限り，テープ・マークが，テープ上の最初のレコードとして書かれる。標準外ラベルおよび追加のユーザー・ラベルは，処理されない。

直接アクセス装置の用法

VERIFYオプション

VERITYオプションは，書出し操作が行なわれるたびにそのあとで読取チェックを行なうことを指定するものである。このオプションは，直接アクセス装置の場合にだけ許される。これは，2321に対して想定される。

EXTENTNUMBERオプション

EXTENTNUMBERオプションは，REGIONALまたはINDEXEDデータ・セットに対して用いられるエクステントの数を指定するものである。

REGIONALデータ・セットの場合は，EXTENTNUMBER（n）は任意選択である。指定する場合は，nは，0より大きく，256より小さいものでなければならない。nの省略時解釈は1である。

INDEXEDデータ・セットの場合は，EXTENTNUMBER（n）は任意選択である。指定するときは，nの値には，すべてのデータ域エクステント，マスター索引およびシリンダー索引エクステント（互いに隣り合っていなければならない），およびすべての独立オーバーフロー・エクステントを含めなければならない。マスター索引とシリンダー索引は，それぞれ別の EXTENT ジョブ制御ステートメントを必要とするが，1つのエクステントとして数える。したがって，指定することのできる最低の数は，2である（1個の主要データ域に対して1つのエクステント，シリンダー索引に対して1つ）。nの最高値は255で，省略時解釈の値は2である。

EXTENTNUMBER オプションの用法のさらにくわしい点については，このコンパイラーのプログラマーの手引を参照のこと。

INDEXMULTIPLEオプション

INDEXMULTIPLE オプションは，INDEXED データ・セットの場合に，このファイルに対してマスター索引が作られるか，または作られたことを指定するためのものである。

HIGHINDEXオプション

HIGHINDEXオプションは，INDEXD データ・セットの場合に，高位のレベルの索引が，MEDIUM オプションで指定された装置タイプとは違う装置に入っているときに その装置のタイプ（2311，2314，2321，3330，または3340 ）を指定するのに用いられる。

OFLTRACKS（n）オプション

OFLTRACKS（n）オプションは，INDEXED 編成の OUTPUTファイルの場合に，追加レコード用に各シリンダーに確保されるトラックの数を指定するために用いられる。指定する数は

CTLASAコード	CTL360コード・バイト		働　き
印刷前の働き	印刷後の働き	印刷を伴なわないときの働き	
+	00000001	−	印刷のみ（スペースなし）
b	00001001	00001011	1行スペース
0	00010001	00010011	2行スペース
−	00011001	00011011	3行スペース
1	10001001	10001011	チャネル　1へスキップ
2	10010001	10010011	チャネル　2へスキップ
3	10011001	10011011	チャネル　3へスキップ
4	10100001	10100011	チャネル　4へスキップ
5	10101001	10101011	チャネル　5へスキップ
6	10110001	10110011	チャネル　6へスキップ
7	10111001	10111011	チャネル　7へスキップ
8	11000001	11000011	チャネル　8へスキップ
9	11001001	11001011	チャネル　9へスキップ
A	11010001	11010011	チャネル10へスキップ
B	11011001	11011011	チャネル11へスキップ
C	11100001	11100011	チャネル12へスキップ

図12.3　IBM 1403および5203印刷制御コード

CTLASAコード	CTL360 コード・バイト	働　き
V	00000001	スタッカー1選択
W	01000001	スタッカー2選択
なし	10000001	スタッカー3選択

図12.4　IBM2540カード読取穿孔装置の制御コード

CTLASAコード	働　き
b	1行スペースして印刷
0	2行スペースして印刷
−	3行スペースして印刷
1	チャネル　1へスキップした後で印刷
2	チャネル　2へスキップした後で印刷
3	チャネル　3へスキップした後で印刷
4	チャネル　4へスキップした後で印刷
5	チャネル　5へスキップした後で印刷
6	チャネル　6へスキップした後で印刷
7	チャネル　7へスキップした後で印刷
8	チャネル　8へスキップした後で印刷
9	チャネル　9へスキップした後で印刷
A	チャネル10へスキップした後で印刷
B	チャネル11へスキップした後で印刷
C	チャネル12へスキップした後で印刷

図12.5　IBM3525カード穿孔印刷装置制御コード（CTLASA）

CTL360コード・バイト	働　き
00001101	1行目に印刷
00010101	2行目に印刷
00011101	3行目に印刷
00100101	4行目に印刷
00101101	5行目に印刷
00110101	6行目に印刷
00111101	7行目に印刷
01000101	8行目に印刷
01001101	9行目に印刷
01010101	10行目に印刷
01011101	11行目に印刷
01100101	12行目に印刷
01101101	13行目に印刷
01110101	14行目に印刷
01111101	15行目に印刷
10000101	16行目に印刷
10001101	17行目に印刷
10010101	18行目に印刷
10011101	19行目に印刷
10100101	20行目に印刷
10101101	21行目に印刷
10110101	22行目に印刷
10111101	23行目に印刷
11000101	24行目に印刷
11001101	25行目に印刷

図12.6　IBM3525カード穿孔印刷装置制御コード（CTL360）

次の範囲にあるものでなければならない。

$0 \leq n \leq 8$　データ・セットが2311上にあるとき

$0 \leq n \leq 17$　データ・セットが3330上にあるとき

$0 \leq n \leq 18$　上記以外の装置上にあるとき

INPUTファイルおよびUPDATEファイルの場合は，レコー

ドが追加されなければ，このオプションは無視される。

印刷／穿孔の制御（CTL360／CTLASA）

印刷／穿孔の制御オプションCTLASAおよびCTL360は，CONSECUTIVEデータ・セットと結びつけられたOUTPUTファイルに対してだけ用いられる。このオプションは，レコードの最初の文字を制御文字として解釈することを指定するものである。

1. CTLASAオプションでは，ANS制御文字を指定する。

2. CTL360オプションでは，IBMシステム／360機械コード制御文字を指定する。

上記のオプションで使用できるコードとその働きが，図12.3，12.4，12.5，12.6，12.7および12.8にリストしてある。

装置からの独立性	ホッパー[1]	CTLASAコード	働き
独立	1	V	スタッカー選択 1
独立	1	W	スタッカー選択 2
独立	2	V	スタッカー選択 5[2]
独立	2	W	スタッカー選択 3
依存	½	V	スタッカー選択 1
依存	½	W	スタッカー選択 2
依存	½	X	スタッカー選択 3
依存	½	Y	スタッカー選択 4
依存	½	Z	スタッカー選択 5[2]

[1] ホッパー選択は，FUNCTIONオプション，またはASSGNステートメント（装置独立のファイルの場合）のどちらかによって行なう。スタッカー選択の如何にかかわらず，2つのホッパーを独立して使用することができる。別のホッパーから入ったカードの間では，競争は行なわれない。（第10章の「ENVIRONMENT属性」を参照のこと。）

[2] 5425を使用しているとき，または使用中の2560がスタッカーを4つしかもたないモデルであるときは，スタッカー4が用いられる。

注： FUNCTION(w)オプションをもつRECORDファイルの場合は，2560で印刷制御文字を使用することができないので，CTLASAまたはCTL360オプションをもつことはできない。

図12.7 IBM2560および5425多能カード処理装置制御コード（CTLASA）

ホッパー[1]	CTL360コード	働き
1	00010011	スタッカー選択 1
1	00100011	スタッカー選択 2
1	00110011	スタッカー選択 3
1	01000011	スタッカー選択 4
1	01010011	スタッカー選択 5[2]
2	10010011	スタッカー選択 1
2	10100011	スタッカー選択 2
2	10110011	スタッカー選択 3
2	11000011	スタッカー選択 4
2	11010011	スタッカー選択 5[2]

[1] ホッパーの選択は，FUNCTIONオプションによって行なわれる。スタッカー選択の如何にかかわらず，2つのホッパーを独立して使用することができる。別のホッパーから入ったカードの間では競争は行なわれない。（第10章の「ENVIRONMENT属性」を参照のこと。）

[2] 5425を使用しているとき，または使用中の2560がスタッカーを4つしかもたないモデルのときは，スタッカー4が用いられる。

注： FUNCTION(w)オプションをもつRECORDファイルの場合は，2560で印刷制御文字を使用することができないので，CTLASAまたはCTL360オプションをもつことはできない。

図12.8 IBM2560および5425多能カード処理装置制御コード（CTL360）

データ交換（COBOL）

このCOBOLオプションは，PL／Iで書かれたプログラムとCOBOLで書かれたプログラムとの間で，データの交換を行なわせるためのものである。このオプションは，ファイルに結びつけられたデータ・セットの中にある構造が，COBOLの算法にしたがってマップされていることを指定している。以下に示すような制約が，COBOLオプションをもつファイルの取扱いに適用される。COBOLのデータ・タイプに対応するPL／Iのデータ・タイプについては，第17章の「言語間の連絡機能」で説明する。

COBOLオプションをもつファイルは，READ INTO，WRITE FROM，およびREWRITE FROMステートメントに対してだけ使用することができる。ファイル名を引数として渡したり，ファイル変数に割り当てたりすることはできない。転送される変数に，添字をつけてはならない。

READステートメントの実行中にON－条件が起ったときに，INTOオプションで指名された変数をオン・ユニットの中で使用することはできない。完了したINTO変数が必要なときは，オン・ユニットから正常に戻ってからにしなければならない。

EVENT オプションは，PL/I と COBOL の両方の構造のマップが全く同じである（つまり，すべての要素項目の限界が同じである）とコンパイラーが判断できる場合にだけ，使用することができる。マップが同じでないとき，または両方が同じであるかどうかコンパイラーにわからないときは，COBOL の算法によってマップされたレベル1の項目を表わす中間的な変数が作られる。PL/I の変数は，WRITE ステートメントが実行される前にこの中間的な変数に割り当てられるか，あるいはREAD ステートメントが実行されたあと，この変数から割り当てられる。

データ管理の最適化（INDEXAREA/NOWRITE/ADDBUFF）

ENVIRONMENT属性のデータ管理の最適化オプションは，INDEXED データ・セットをアクセスするとき，ある特定の環境の場合にプログラムの効率を上げるためのものである。

INDEXAREAオプションは，INPUT または UPDATE に対して開かれた DIRECT ファイルの転送の速度を上げるためのもので，このオプションがあると，シリンダー索引の全体または一部が主記憶域に入れられる。割り振るサイズは，次のものでなければならない。

(m ＋ 3)*(キーの長さ ＋ 6)

ここで'm'は，主記憶域の中に入れる索引エントリーの数である。（1つの索引エントリーはそれぞれ，1シリンダーのデータに対応している）。すべての索引が主記憶域に収容される場合は，'m'の値はデータ・シリンダーの数より1だけ大きいものでなければならない。

NOWRITEオプションは，DIRECT UPDATEファイルに対して使用される。このオプションは，このデータ・セットにはレコードが追加されないこと，およびレコードの追加にだけ関係のあるデータ管理モジュールは不要であることをコンパイラーに知らせる働きをする。この結果，目的プログラムのサイズを減らすことができる。

ADDBUFFオプションは，DIRECTファイルの場合に，新しいレコードを追加するとき，データ・セット上のレコードを並べ直すための作業スペースとして使用できる内部記憶域の量を指定するのに用いられる。指定できる記憶域の最低量は，次の式によって決まる。

m(ブロック・サイズ ＋ キーの長さ ＋ 50) ＋ 24

ここで，'m'は，一時点に記憶域に読み込まれるブロックの数である。指定可能な最適のブロック数は，直接アクセス装置の1トラックにいっぱいになるものである。

キーの分類

GENKEY（総称キー）オプションは，INDEXEDおよびキー順のVSAM データ・セットに対してだけ用いられる。このオプションを用いると，プログラマーは，データ・セットの中に記録されたキーを分類し，そのキーのクラスにしたがってレコードをアクセスするためにSEQUENTIAL KEYED INPUT または SEQUENTIAL KEYED UPDATE ファイルを使用することができるようになる。

総称キーは，キーのクラスを識別する文字ストリングである（そのストリングで始まるすべてのキーは，そのクラスのメンバーである）。たとえば，記録されたキー'ABCD'，'ABCE'，および'ABDF'はすべて，総称キー'A'および'AB'によって識別されるクラスのメンバーであるが，最初の2つは，クラス'ABC'のメンバーでもある。また，3つの記録されたキーは，それぞれ，クラス'ABCD'，'ABCE'，および'ABDF'の唯一のメンバーであると考えることができる。

GENKEYオプションを使用すると，プログラマーは，特定クラスのキーをもつ最初のレコードから，INDEXED データ・セットの順次的な読取りまたは更新を開始することができる。この場合，そのクラスは，READ ステートメントのKEYオプションの中にその総称キーを含めることによって識別される。（特定のクラスのキーをもつ最初のレコードは，READ KEYによって取り出すことができるが，KEYオプションと同じステートメントの中で KEYTO オプションを使用することができないので，実際のキーは，レコードの中にキーが内蔵されていない限り，取り出すことはできない。）後読のレコードは，KEYオプションをもたない READ ステートメントによって，読むことができる。キー・クラスの終りになっても，表示は行なわれない。

次の例では，キーの長さは4バイト以上とする。

```
DCL IND FILE RECORD SEQUENTIAL KEYD
    UPDATE ENV (INDEXED GENKEY…);
        ⋮
READ FILE(IND) INTO(INFIELD) KEY ('ABC');
        ⋮
NEXT:  READ FILE (IND) INTO (INFIELD);
        ⋮
       GO TO NEXT;
```

最初のREAD ステートメントは，データ・セットの中にある，'ABC'で始まるキーをもつ最初のレコードを INFIELD に読み込む働きをする。そのあと，2番目のREAD ステートメントが実行されるたびに，次に高いキーをもつレコードが取り出される。2番目のREAD ステートメントを繰り返して実行してゆくうちに，キー・クラスの終りになっても表示は行なわれないので，より高いキー・クラスからレコードを読むこともできる。そのキー・クラス

の範囲外のものを読みたくない場合は，プログラマーの責任において，各キーをチェックしなければならない。のちに，最初のREADステートメントを実行すると，データ・セットはつねに，キー・クラス 'ABC' の最初のレコードに再び位置づけされる。

指定されたクラスのキーをもつレコードがデータ・セットの中にない場合は，KEY条件が起こる。このあと，データ・セットは，それより高いキーをもつ次のレコードか，またはファイルの終りのどちらかに位置づけされる。

KEYLENGTHオプションで指定された長さよりも短い原始キーを指定しているREAD ステートメントが実行されるとき，そこにGENKEY オプションがあるかないかによってどのような効果の違いがあるか注意していただきたい。GENKEY があれば，キーは総称キーとして解釈され，データ・セットは，その原始キーで始まるキーをもつ，データ・セット内の最初のレコードに位置づけされる。GENKEY オプションが指定されていなければ，短い原始キーは，指定されたキーの長さになるよう右側にブランクが補充され，データ・セットは，この補充されたキーをもつレコードに位置づけされる（このようなレコードが存在していれば）。

GENKEYオプションを使用しているとき，指定されたキーの長さと同じかそれより長い原始キーを与えてもその結果に影響はない。この場合，原始キーは，必要に応じて右側で切り捨てられ，指定のレコードを識別する（このレコードのキーは，そのクラスの唯一のメンバーであると考えられる）。

長さの変わるストリング・オプション（SCALARVARYING）

SCALARVARYING オプションは，長さの変わるストリングの入出力の場合に用いられる。長さの変わるストリングを位置指定モードによって転送できるのは，このオプションを使用しているときだけである。ストリングが転送されるときに長さの変わる要素ストリングに2バイトの長さ接頭語を含めることによって（あるいはそれを認識することによって），このようなことが可能になる。

長さの変わるストリングに記憶域を割り振るとき，コンパイラーは，ストリングの現在の長さを指定する2バイトの接頭語をその中に含める。長さの変わる要素ストリングの場合，SCALARVARYING がそのファイルに対して指定されているときだけ，この接頭語が含められたり（出力のとき），認識されたりする（入力のとき）。

長さの変わる要素ストリングをもつデータ・セットの作成および読取りに位置指定モードのステートメント（LOCATEおよびREAD SET）を用いるときは，緩衝域の位置を指定するポインターはつねに長さ接頭語の先頭を指すとみなされるので，長さ接頭語が存在していることを表わすために，SCALARVARYING を指定しなければならない。

注：

1. ファイルに対してSCALARVARYING が指定されている場合は，それに書き出したり，それから読んだりすることができるのは，構造形式でなく，次元ももたない，レベル1のスカラーの，長さの変わるストリングだけである。プログラマーは，レコードの長さを決めるとき，この長さ接頭語を入れるために2バイトの余裕をみておかなければならない。

2. SCALARVARYING を用いて作成されたデータ・セットは，SCALARVARYING を指定しているファイルによってのみアクセスしなければならない。

3. SCALARVARYING とCTLASA／CTL360 の両方を同じファイルに対して指定してはならない（このようにすると，最初のデータ・バイトが不明確になる）。

4. PL／I DまたはPL／I Fコンパイラーによって作成されたデータ・セットに対して，SCALARVARYING を使用してはならない。これらのコンパイラーでは，長さ接頭語の作成も認識も行なわれない。

キーの長さオプション（KEYLENGTH）

KEYLENGTHオプションは，KEYED ファイルに対して，記録されたキーの長さを指定するものである。INDEXED またはREGIONAL（3）ファイルの場合は，KEYLENGTHを必ず指定しなければならない。

キーの位置オプション（KEYLOC）

KEYLOC オプションは，INDEXED データ・セットを作成するときに，レコードの中の内蔵されたキーの先頭の位置を指定するために使用することができる。非ブロック化レコードの場合は，KEYLOCは任意選択であるが，ブロック化レコードの場合は，必ずKEYLOC を指定しなければならない。その位置は，次の範囲内のものでなければならない。

$1 \leq n \leq$ レコードサイズ － キーの長さ ＋1

つまり，キーは，レコードより大きいものであってはならず，レコード内に完全に収まるものでなければならない。

VSAMデータ・セット

VSAM データ・セットはつねに，直接アクセス装置上にあり，キー順またはエントリー順の編成になっている。これらの編成は通常，PL／IのINDEXEDおよびCONSECUTIVE編成にそれぞれ対応している。VSAM データ・セットに結びつけられるファイ

オ プ シ ョ ン	キー順(INDEXED)	エントリー順(CONSECUTIVE)
RECSIZE	開かれるときにチェックされる	開かれるときにチェックされる
BLKSIZE	無視される	無視される
COBOL	INDEXEDの場合	CONSECUTIVEの場合
VERIFY	無視される	無視される
EXTENTNUMBER	無視される	適用されない
INDEXMULTIPLE	無視される	適用されない
HIGHINDEX	無視される	適用されない
OFLTRACKS	無視される	適用されない
INDEXAREA	無視される	適用されない
NOWRITE	無視される	適用されない
ADDBUFF	無視される	適用されない
GENKEY	INDEXEDの場合	適用されない
SCALARVARYING	INDEXEDの場合	CONSECUTIVEの場合
KEYLENGTH	開かれるときにチェックされる	適用されない
KEYLOC	開かれるときにチェックされる[1]	適用されない
BUFFERS	無視される	無視される
レコード形式	無視される	無視される
PASSWORD	VSAMによってチェックされる	VSAMによってチェックされる

[1] キー順のVSAMデータ・セットはすべて，内蔵されたキーをもっている。INDEXEDデータ・セットから変換されたVSAMデータ・セットをプログラムが使用しており，しかも，元のプログラムが，内蔵されないキーをもつよう設計されている場合は，ENVIRONMENT(VSAM)を指定してはならない。

図12.9 VSAMデータ・セットに対して使用された場合のENVIRONMENTオプションの効果

ルの宣言は，ENVIRONMENT(VSAM)を指定する点以外は，本質的には対応するINDEXEDまたはCONSECUTIVE ファイルの場合と同じである。キー順のVSAMデータ・セットに対してENV(INDEXED)が指定されていると，オペレーティング・システムによって自動的にISAMインターフェースが使用される。このコンパイラーのプログラマーの手引書に，VSAMデータ・セット編成に関する詳細，JCLを通じて使用できる機能，PL/Iファイルを通してVSAMデータ・セットをアクセスする方法，などが記載されている。

ファイルの宣言と開き方

最初にVSAMデータ・セットを開く前に，DEFINEユーティリティー・プログラムによって，データ・セットの構造(レコードおよびキーの長さなど)がシステムに対して定義される。したがって，データ・セットの構造に影響を与えるENVIRONMENTオプションの多くは，定義された値と矛盾している場合をのぞいて，無視される。矛盾しているときは，UNDEFINEDFILE条件が起こる。

図12.9は，ディスク・データ・セットがVSAMデータ・セットに対して使用されるときに適用されるすべてのENVIRONMENTオプションの影響を要約したものである。

INDEXED 編成の場合には許されない2つの属性の組合せが，キー順のVSAMデータ・セットに結びつけられるファイルの場合には許されている。これらの組合せとは，直接アクセスでの入力に位置指定モードを使用するための，BUFFEREDとDIRECTの組合せ，および順次アクセスでEVENTオプションを使用するための(任意の一時点では，1つの操作だけを未解決の状態にすることができる)，UNBUFFEREDとSEQUENTIALの組合せである。

エントリー順のデータ・セットと結びつけられるファイルの場合に許される属性は，CONSECUTIVE編成の場合と同じである。

操　作

INDEXEDおよびCONSECUTIVE 編成の場合に有効なステートメントはすべて，対応するVSAMデータ・セットの場合にも有効である。さらに，INDEXEDの場合には有効でないステートメントの中にも，キー順のデータ・セットの場合には有効なものがある。

図12.13は，キー順のVSAMデータ・セットで使用できるステートメントとオプションを要約したものである。エントリー順のVSAMデータ・セットについては，図12.10を参照のこと。

ASCIIデータ・セット

ASCII を用いる磁気テープ上のデータ・セットを，PL/Iで作成またはアクセスすることができる。このコンパイラーでは，F，FB，U，D，およびDBレコード形式がサポートされている。

F，FB，およびU形式は，他のデータ・セットの場合と同じように扱われる。DおよびDB形式（他のデータ・セットの場合のVおよびVB形式に相当する）については，次に説明する。

レコード形式のほかに，2つのENVIRONMENTオプション（ASCII，および緩衝域オフセット・オプションBUFOFF）を指定することができる。

ASCII データ・セットに書くことができるのは，文字データだけである。したがって，このデータ・セットを作成するときは，文字ストリング変数から転送しなければならない。この変数は，CHARACTERと同様，VARYING 属性をもつことができるが，長さの変わる文字ストリングの長さバイト（2バイト）を転送することはできない。いいかえれば，長さの変わる文字ストリングは，SCALARVARYING ファイルを使用している ASCII データ・セットに転送することはできない。また，長さの変わるストリングが含まれているデータ集合体も，転送することができない。

ASCII データ・セットは磁気テープ上になければならないので，これは当然CONSECUTIVE編成のものでなければならない。結びつけられるファイルは，緩衝域をもっていなければならない。

ASCIIオプション

このオプションは，データ・セット上のデータを表わすのに用いられるコードがASCII であることを指定するものである。

BUFOFFオプションおよびブロック接頭語フィールド

ASCII データ・セットの各ブロックの先頭に，ブロック接頭語フィールドと呼ばれるフィールドをつけることができる。この長さは，1から99バイトまでのいずれでもよい。この緩衝域オフセット・オプションは，このフィールドの長さをデータ管理プログラムに表示する働きをするので，データ・セットのアクセスまたは作成は，各物理ブロックの先頭からのこのオフセットの位置で開始される。PL／Iは，このフィールドへのアクセスはサポートしておらず，また，システム／360およびシステム／370で使用される情報も含んでいない。このブロック接頭語に入っている情報をデータ管理プログラムが使用するのは，非ブロック化またはブロック化可変長レコード（つまり，D－またはDB－形式レコード）の場合に，ブロックの長さを記録するためにブロック接頭語フィールドを使用できる場合だけである。この場合，このフィールドは4バイトの長さで，ブロック接頭語フィールド自体も含めてブロックの長さをバイト単位で示す，右づめの10進数文字値を含んでいる。これは，ブロックの長さフィールドと全く同じになる。

緩衝域オフセット・オプションの形式はBUFOFF（n）である。接頭語の長さを表わす数値は，"n"として指定しなければならない。これは，10進整定数またはFIXED BINARY(31,0) STATIC という属性をもつ変数として指定することができる。その最小値はゼロで，最大値は99である。接頭語の長さの指定がないということは，ブロック接頭語がブロックの長さフィールドとして用いられること，つまりこのフィールドが4バイトであることを意味している。ブロックの長さは，データ管理プログラムによって，接頭語に挿入される。

入力の場合，ASCII データ・セットは，1から99バイトまでの長さのブロック接頭語フィールドをもっているか，またはブロック接頭語フィールドを全くもっていないときは，アクセスすることができる。これは，ブロック接頭語フィールドがブロックの長さフィールドとして使用されているかどうかに関係なくアクセス可能である。出力の場合，正しいレコード形式のいずれかを使用しているデータ・セットは，ブロック接頭語をつけずに作成することができるが，ブロック接頭語の作成がPL／Iでサポートされるのは，それがブロックの長さフィールドとして使用できるときだけである。

BUFOFFオプションは，ASCIIデータ・セットの場合にだけ使用することができる。

D－形式およびDB－形式レコード

各レコードは，それぞれ長さが違っていてもよい。この2つの形式は以下の通りである。

D－形式：　レコードはブロック化されておらず，1つのレコードが1つのブロックを構成している。各レコードは，次のものから構成されている。

4バイトの制御バイト

データ・バイト

4バイトの制御バイトには，レコードの長さが入る。この値はデータ管理プログラムによって挿入されるので，プログラマーは何も処置をとる必要がない。さらに，ブロックの先頭にブロック接頭語フィールドがあってよく，ここには，ブロックの長さを入れておくことができる。

DB－形式：　レコードはブロック化されている。この点以外についてD－形式のところで述べたことが，そのままDB－形式にもあてはまる。

省略時解釈の規則

「レコード形式の省略時解釈」のところで述べた規則のほかに，次のような規則も適用される。

ENVIRONMENTオプションの中でASCIIは指定されていないが，BUFOFF，D，またはDBのいずれかが指定されているという場合は，ASCIIとみなされる。

CONSECUTIVE編成

CONSECUTIVE編成のデータ・セットでは，レコードはキーをもっていない。データ・セットが作成されるとき，レコードは提示された順に連続的に書き出される。レコードは，書き出された順，あるいはBACKWARDS属性を使用しているときはその逆の順でだけ取り出すことができる。したがって，結びつけるファイルは，SEQUENTIAL属性をもつものでなければならない。
CONSECUTIVEデータ・セットには，F－形式，FB－形式，V－形式，VB－形式，D－，DB－，またはU－形式レコードを収容することができる。データ・セット内のレコードがV－，VB－，D－またはDB－形式の場合は，BACKWARDS属性を指定することはできない。

ENVIRONMENT属性のCONSECUTIVEオプションとSEQUENTIAL属性の違いに注意していただきたい。CONSECUTIVEは，データ・セットの物理的な構成を指定するのに対して，SEQUENTIALは，ファイルをどのように処理するかを指定するものである。CONSECUTIVE編成のデータ・セットは，SEQUENTIALファイルと結びつけなければならないが，INDEXEDまたはREGIONAL 編成のデータ・セットは，SEQUENTIALまたはDIRECTファイルのどちらとでも結びつけることができる。

磁気テープ上のCONSECUTIVEデータ・セットは，前向きまたは後向きに読むことができる。データ・セットを後向きに読むときは，結びつけられるファイルは，BACKWARDS属性をもっていなければならない。最初にデータ・セットを前向きに読むか書くかしてから，同じプログラムで後向きに読むという場合は，ファイルが閉じられるとき（または複数ボリューム・データ・セットでは，ボリューム切替えが行なわれるとき）に，通常の巻戻しを行なわないようにするために，ENVIRONMENT属性の中でLEAVEオプションを指定しなければならない。可変長レコードのデータ・セットを後向きに読むことはできない。

CONSECUTIVEデータ・セットが作成されたあと，それをアクセスするファイルは，SEQUENTIAL INPUT，OUTPUT，またはUPDATE（これは，磁気テープ・データ・セットの場合をのぞく）として開くことができる。（OUTPUTとして開かれたときは，データ・セットは前のものを消して書かれる。）図12.6は，CONSECUTIVEデータ・セットの作成とアクセスに使用できるデータ転送ステートメントとオプションをリストしたものである。

SEQUENTIAL UPDATE

CONSECUTIVEデータ・セットがSEQUENTIAL UPDATEファイルによってアクセスされるときは，REWRITEステートメントによって更新できるようにするために，前もってレコードをREADステートメントによって取り出しておかなければならない。しかし，取り出したレコードをすべて書き直す必要はない。REWRITEステートメントはつねに，読まれた最後のレコードを更新する。

次の例について考えてみよう。

```
READ FILE(F) INTO(A);
      :
READ FILE(F) INTO(B);
      :
REWRITE FILE(F) FROM(A);
```

このREWRITEステートメントは，2番目のREADステートメントによって読まれたレコードを更新する。最初のステートメントによって読まれたレコードは，2番目のREADステートメントが実行されたあとで書き直すことはできない。

INDEXED編成

INDEXED編成のデータ・セットは，直接アクセス装置上になければならない。そのレコードは，F－またはFB－形式レコードでなければならず，各レコードに結びつけられるキーにしたがって論理的な順序で並べられている。キーは，各レコードを唯一無二のものとして識別できる文字ストリングである。論理レコードは，システム／360およびシステム／370の照合順序にしたがって，キーの昇順にデータ・セットの中に並べられる。オペレーティング・システムのデータ管理ルーチンは，キーが与えられたとき，データ・セットの中に組み込まれているキーを使用して，レコードの位置を探し当てる。

CONSECUTIVE編成とは異なり，INDEXED編成の場合は，すべてのレコードを順次方式でアクセスする必要はない。INDEXEDデータ・セットは，順次的に作成しなければならないが，一旦作成したあとは，結びつけられるファイルのもつ属性は，SEQUENTIALまたはDIRECT，あるいはINPUTまたはUPDATEのいずれであってもよい。ファイルがDIRECT属性をもっているときは，レコードの取出し，追加，および置換えをランダムに行なうことができる。

図12.11は，INDEXEDデータ・セットの作成およびアクセスに使用することのできるデータ転送ステートメントとオプションをリストしたものである。

キー

キーには，記録されたキーと原始キーの2種類がある。記録された

キーは，そのレコードを識別するためにデータ・セット内の各レコードに実際に出てくる文字ストリングである。この長さは，255桁を越えてはならない。原始キーは，データ転送ステートメントのKEYまたはKEYFROMオプションの中に出てくる式の文字ストリング値であり，そのステートメントが参照しているレコードを識別するために用いられる。INDEXEDデータ・セットを直接アクセスする場合は，転送ステートメントにそれぞれ原始キーを含めなければならない。

INDEXED データ・セットの中の記録されたキーの長さは，ENVIRONMENTのKEYLENGTHオプションによって定義する。原始キーの長さが，記録されたキーに対して指定された長さより大きいときは，原始キーは右側で切り捨てられる。原始キーが，指定されたキーの長さより短く，しかもGENKEYオプションが使用されていないときは，原始キーは，指定された長さに合わせて右側にブランクが補われる。

INDEXED データ・セットの記録されたキーは，論理レコードの中に内蔵してもよいし，別にしてもよい。キーをレコードの中に内蔵する場合は，レコード内におけるキーの位置を与えるために，ENVIRONMENTのKEYLOC(n)オプションを指定しなければならない。

内蔵されたキー

レコードがブロック化されているINDEXEDデータ・セットは，ゼロより大きい値が指定されたENVIRONMENT属性のKEYLOCオプションをもっていなければならない。KEYLOCオプションは，内蔵されたキーのレコード内における絶対的な位置を指定するものであることに注意していただきたい。

レコードがブロック化されていないINDEXEDデータ・セットは，レコードが，内蔵されたキーをもっているときだけ，KEYLOCオプションを指定しなければならない。KEYLOCが指定されていないか，またはゼロのときは，そのレコードは，内蔵されたキーをもたないものとみなされる。

内蔵されたキーを使用すると，順次入力中にKEYTOオプションを使用する必要がなくなるが，出力の場合，KEYFROMオプションはやはり必要である。(しかし，KEYFROMオプションによって指定されるデータが，レコード変数の内蔵されたキー部分そのものであっても構わない。)レコードがブロック化されていないデータ・セットの場合は，内蔵されたキーがある場合でも，各レコードの前に別個の記録されたキーがつけられる。したがって，非ブロック化レコードの場合は，キー情報のための追加スペースが必要である。レコードがブロック化されている場合は，各ブロックの最後のレコードのキーだけが，ブロックの前に別に記録される。

内蔵されたキーをもつデータ・セットにレコードを追加するためのLOCATEまたはWRITEステートメントの実行中には，KEYFROMオプションの中の式の値が，レコード変数の中の内蔵されたキーの位置に割り当てられる。レコード変数は，構造メンバーとして宣言された内蔵されたキーをもつ構造として宣言することができるが，このような内蔵されたキーは，長さの変わるストリングとして宣言してはならないことに注意していただきたい。

データ・セットの作成

INDEXED データ・セットを作成するときは，結びつけるファイルはSEQUENTIAL OUTPUT として開かなければならず，また，キーの値の昇順にレコードを提示しなければならない。(キーの順序が間違っているときは，KEY条件が起こる。) INDEXED データ・セットを作成するためにDIRECTファイルを使用することはできない。

INDEXED データ・セットを作成したあと，それをアクセスするファイルは，SEQUENTIAL INPUTまたはUPDATEとして，あるいはDIRECT INPUTまたはUPDATE として開くことができる。また，データ・セットの終りにレコードを追加するために，OUTPUTとして開くこともできる。このようなレコードのキーは，そのデータ・セットの既存のキーよりも高い値をもっていなければならず，また，昇順に並んでいなければならない。出力として開くときに，データ・セットに割り振られる記憶域を増やすこともできる。

順次アクセス

INDEXED データ・セットをアクセスするのに用いられるSEQUENTIALファイルは，INPUTまたはUPDATE属性のどちらかを指定して開くことができる。データ転送ステートメントに原始キーを含める必要もないし，またファイルがKEYED属性をもっている必要もない。順次アクセスは，記録されたキーの値の昇順に行なわれる。つまり，レコードは上記の順に取り出されるのであって，必ずしも，データ・セットに入れられた順番ではない，ということである。

INDEXED データ・セットをアクセスするSEQUENTIAL UPDATE ファイルに対するREADとREWRITEの両ステートメントの関係についての規則は，EVENTオプションを使用できない点をのぞけば，CONSECUTIVEデータ・セットの場合(上述)と同じである。

更新されるレコードの中にある，内蔵されたキーを変更してはならない。修正されたレコードはつねに，データ・セットの中の更新されるレコードの上に書き出される。

INDEXED データ・セットを順次アクセスしているときに，READステートメントのKEYオプションの中で原始キーを与え

ることによって，特定のレコードを読み，その地点から，データ・セットの順次的な読取りをつづけることができる。（結びつけられたファイルは，KEYED属性をもっていなければならない。）したがって，KEYオプションを含むREADステートメントは，そのキーが与えられているレコードを読むことになる。該当するレコードが存在していないと，KEY条件が起こる。KEYオプションをもたない後続のREADステートメントは，次に高い記録されたキーをもつレコードを読む。

直接アクセス

INDEXEDデータ・セットをアクセスするために用いられるDIRECTファイルは，INPUTまたはUPDATE属性のどちらかをつけて開くことができる。データ転送ステートメントはすべて，原始キーをもっていなければならない（DIRECT属性は，KEYED属性を暗示しているため）。

DIRECT UPDATEファイルは，INDEXEDデータ・セットの中のレコードの取出し，追加，または置換えのために使用することができる。レコードの置換えには，次の規則が適用される。

REWRITEステートメントの中の原始キーによって指定されたレコードが，新しいレコードによって置き換えられる。データ・セットに含まれるレコードがFB－形式の場合は，前の入出力ステートメントが，置き換えられるレコードを読んだREAEステートメントでない限り，REWRITEスレートメントでレコードが置き換えられると，READステートメントが暗示的に実行される。

REGIONAL編成

REGIONAL編成のデータ・セットは，区域番号によって識別される区域に分割されており，そのそれぞれの区域には，REGIONAL編成のタイプの如何によって，1個のレコードまたは数個のレコードを入れることができる。区域は，ゼロから始まる連続番号をもっており，データ転送ステートメントの中でその区域番号（さらに大抵の場合はキーも）を指定することによって，レコードをアクセスすることができる。

データ・セットをREGIONAL編成にすると，プログラマーは，データ・セット内におけるレコードの物理的な位置を制御することができる。

REGIONALデータ・セットは，CONSECUTIVEまたはINDEXEDデータ・セットの場合と同じように，区域番号の昇順にレコードを提示して作成することができる。かわりに，直接アクセスを使用することもできるが，この場合は，レコードをランダムな順序で提示して，前もって形式設定された区域に直接挿入することができる。一旦REGIONALデータ・セットが作成されたあとは，SEQUENTIALまたはDIRECT，およびINPUTまたはUPDATE属性をもつファイルによって，アクセスすることができる。データ・セットがSEQUENTIAL INPUTまたはSEQUENTIAL UPDATEファイルと結びつけられている場合は，区域番号もキーも指定する必要がないことに注意していただきたい。ファイルがDIRECT属性をもっている場合は，レコードの取出し，追加，置換えをランダムに行なうことができる。

図12.12は，REGIONALデータ・セットの作成およびアクセスに使用できるデータ転送ステートメントとオプションをリストしたものである。

キー

キーには，記録されたキーと原始キーの2種類がある。記録されたキーは，データ・セット内の各レコードの直前にあり，そのレコードを識別するための文字ストリングである。原始キーは，データ転送ステートメントのKEYまたはKEYFROMオプションの中にある式の文字ストリング値であり，そのステートメントが参照するレコードを識別するものである。REGIONALデータ・セットの中のレコードをアクセスするときは，原始キーによって区域番号が与えられる。記録されたキーが与えられることもある。

REGIONAL(3)データ・セットの記録されたキーの長さは，結びつけられたファイルのENVIRONMENT属性のKEYLENGTHオプションによって定義される。INDEXEDデータ・セットのキーとは違って，REGIONALデータ・セットの記録されたキーは，レコードの中に内蔵されているときは，認識されない。

REGIONAL編成のタイプ

REGIONAL編成には，次の2つのタイプがある。

1. REGIONAL(1)データ・セットには，記録されたキーをもたないF－形式レコードが入っている。データ・セット内の各区域には，レコードは1つしか入れられない。したがって，各区域の番号は，データ・セット内における相対的なレコード位置に対応している。

2. REGIONAL(3)データ・セットには，記録されたキーをもつF－形式またはU－形式のレコードが入っている。データ・セット内の各区域は，直接アクセス装置上の1トラックに相当し，通常は1個または数個のレコードを収容することができる。REGIONAL(3)データ・セットを直接アクセスするときは，所要の区域を探し当てるために，原始キーの中で指定された区域番号が用いられる。区域が探し当てられたあとで，レコードを追加するスペース，または原始キーで与えられたものと同じの記録されたキーをもつレコードを

探すために，順次的な探索を行なうことができる。

REGIONAL(1)編成

REGIONAL(1)データ・セットでは，記録されたキーがないので，特定のレコードを識別するための唯一の標識は，区域番号である。原始キーの文字ストリング値は，符号なしの10進整定数(16777215を越えないもの)を表わすものでなければならない。区域番号がこの数より大きいときは，16777216の剰余モジュロが用いられる。たとえば，16777226は10とみなされる。原始キーの中では，0から9までの文字とブランクだけが有効である。先行ブランクはゼロと解釈されるが，数の間にブランクがあってはならない。もしあると，間にはさまったブランクが最初にみつかったところで，区域番号は終りになる。キーの中の文字が8桁より大きいときは，右側の8桁だけが区域番号として用いられる。8桁より小さいときは，左側にブランク(ゼロとみなされる)がとられる。

見かけレコード

REGIONAL(1)データ・セット内のレコードは，有効なデータを含む実際のレコードまたは見かけレコードのどちらかである。REGIONAL(1)データ・セットの中の見かけレコードは，その最初のバイトに定数(8)'1'B があることによって識別される。このような見かけレコードは，作成されるときにデータ・セットに自動的に挿入されるが，データ・セットが読まれるときには無視されないので，それらを認識できるよう，PL/I プログラムを準備しなければならない。見かけレコードを，有効なデータで置き換えることができる。

プログラマーが最初のバイトに(8)'1'B を挿入した場合は，その中の見かけレコードを取り出さないようなデータ・セットにファイルを複写すると，そのレコードは失なわれるので，注意していただきたい。

REGIONAL(1)データ・セットの作成

REGIONAL(1)データ・セットは，順次的に，または直接アクセスによって作成することができる。

データ・セットを作成するためにSEQUENTIAL OUTPUTファイルを使用している場合は，ファイルを開くと，データ・セット内のすべてのトラックがクリアーされ，そのトラック上の使用可能なスペースの量を記録するために各トラックの先頭に容量レコードが作成される。レコードは，区域番号の昇順に提示しなければならない。この順序の中で省略された区域は，見かけレコードで埋められる。この順序に誤りがあったとき，またはキーが重複して提示された場合は，KEY条件が起こる。ファイルが閉じられるとき，データ・セットの終りに残ったスペースは，見かけレコードで埋められる。

緩衝域をもつファイルを用いてデータ・セットが作成されている場合に，ファイルが閉じられる前に出された最後のWRITE または LOCATE ステートメントが，そのデータ・セットの限界を越えてレコードを出力しようとすると，CLOSE ステートメントによってエラー条件が引き起されることがある。

データ・セットを作成するためにDIRECT OUTPUTファイルが用いられる場合は，ファイルが開かれるとき，データ・セット全体が見かけレコードで埋められる。

REGIONAL(1)データ・セットが作成されたあと，それをアクセスするためのファイルは，SEQUENTIAL INPUTまたはUPDATE，あるいはDIRECT INPUTまたはUPDATE として開くことができる。既存のデータ・セットを消してその上に書くときだけは，OUTPUT として開くことができる。

順次アクセス

REGIONAL(1)データ・セットを処理するのに用いられるSEQUENTIALファイルは，INPUTまたはUPDATE属性のどちらかをつけて開くことができる。データ転送ステートメントに，KEYオプションを含めることはできないが，KEYTO オプションを使用することはできるので，ファイルは KEYED 属性をもつことができる。KEYTO オプションで指定されている文字ストリング変数の文字が，8桁を越えているときは，返される値(区域番号)は，左側にブランクが補なわれる。8桁より少ないときは，左側で切り捨てられる。

順次アクセスは，区域番号の昇順に行なわれる。見かけであるか，実際のレコードであるかには関係なくすべてのレコードが取り出されるので，見かけレコードを認識できるように，PL/I プログラムを準備しておかなければならない。

REGIONAL(1)データ・セットをアクセスするSEQUENTIAL UPDATEファイルに対するREADとREWRITEの両ステートメントの関係についての規則は，CONSECUTIVEデータ・セットに対する規則と同じである(上で述べた通り)。

直接アクセス

REGIONAL(1)データ・セットを処理するのに用いられるDIRECTファイルは，INPUTまたは UPDATE 属性のどちらかをつけて開くことができる。すべてのデータ転送ステートメン

トは，原始キーを含んでいなければならない（DIRECT 属性はKEYED 属性を暗示しているため）。

REGIONAL（1）データ・セットの中のレコードの取出し，追加，または置換えをするために，次の規則にしたがって，DIRECT UPDATEファイルを使用することができる。

1. 取出し： 見かけであるか，実際のレコードであるかに関係なく，すべてのレコードが取り出される。見かけレコードを認識できるようにプログラムを準備しなければならない。

2. 追加： 原始キーによって指定された区域内の既存のレコード（実際または見かけの）が，WRITE ステートメントによって新しいレコードと置き換えられる。

3. 置換え： REWRITE ステートメントの中の原始キーによって指定されたレコードが，見かけであるか実際のレコードであるかには関係なく，置き換えられる。

REGIONAL（3）編成

REGIONAL（3）データ・セットの場合は，各レコードは，レコードの直前にある記録されたキーによって識別される。他のレコードから相対的にみた，データ・セット内のレコードの実際の位置は，その記録されたキーによって決まるのではなく，レコードをデータ・セットに追加するためのWRITE ステートメントの原始キーの中で与えられた区域番号によって決まる。

各区域番号は，そのデータ・セットが入っている直接アクセス装置上の1トラックを識別するものであり，16777215 を越えていてはならない。

1つの区域には，1個または数個のレコードを入れることができる。

データ・セットには，F－形式，またはU－形式レコードを入れることができる。データ・セットが前もって形式設定されている場合は，データ・セット内のすべてのトラックはクリアーされており，オペレーティング・システムによって，各トラックの先頭に容量レコード（そのトラック上で使用可能なスペースの量が記録されている）がつけられている。

レコードがデータ・セットに追加されるとき，レコードは，指定された区域が含まれているトラックの最初の使用可能なスペースに，記録されたキーといっしょに書き出される。レコードが読まれるとき，該当する記録されたキーをもつレコードの探索は，指定された区域を含むトラックの先頭から開始され，そのトラックの終りまで続けられる。

原始キー

原始キーの文字ストリング値は，2つの論理的な部分（区域番号と比較キー）をもつと考えることができる。出力のとき，比較キーは記録されたキーとして書かれる。入力のとき，このキーが，記録されたキーと比較される。

原始キーの右端の8文字は，区域番号を表わしており，16777215を越えない，符号なしの10進整定数でなければならない。区域番号がこの数より大きいときは，16777216 のモジュロとして扱われる。つまり，16777226は10として扱われる。区域の指定には，0から9までの文字とブランク文字しか使うことができない。先行のブランクはゼロと解釈される。また，この数の間にブランクを入れることはできず，もし入れると，最初のブランクがあったところで区域番号は終りになる。比較キーは，原始キーの左側の部分を占める文字ストリングであり，区域番号と重複していても，別になっていてもよく，他の有効でない文字によって，区域番号と区別することができる。比較キーの長さは，ENVIRONMENT属性のKEYLENGTHオプションによって指定される。指定されたキーの長さよりも原始キーの方が短い場合は，右側にブランクが補充されて拡張される。レコードを取り出すためには，比較キーと，レコードの記録されたキーとが完全に一致していなければならない。比較キーに，区域番号を含めることができるが，このような場合，原始キーと比較キーは同じである。かわりに，原始キーの一部分だけを使用することはできない。比較キーはつねにKEYLENGTH と同じである。原始キーがKEYLEN＋8 より長いときは，比較キーと区域番号の間にある原始キーの文字は無視される。

原始キーに関する次の例を考えてみよう（文字"b"は，ブランクを表わしている）。

```
KEY ('JOHNbDOEbbbbbb12363')
```

右端の8文字は，区域の指定，すなわちレコードの相対番号を表わしている。関連するENVIRONMENT属性は，KEYLENGTH（14）というオプションをもっているとしよう。レコードを取り出すときは，区域番号 12363 が含まれるトラックの先頭から探索が始まり，JOHNbDOEbbbbbb という記録されたキーをもつレコードが発見されるまで，探索がつづけられる。

オプションが，KEYLENGTH（19）であったとしても，探索は同じ位置から始められる。しかし，比較キーと原始キーは同じ長さなので，この場合の探索は，記録されたキー 'JOHNbDOEbbbbbb12363' をもつレコードに対するものになる。

KEY ('JOHNbDOEbbbbbbDIVISIONb23bb bb34627')

この例では，右端の８文字に先行ブランクが含まれているが，これはゼロと解釈される。探索は，区域番号 00034627 から始まる。KEYLENGTH(14) が指定されている場合は，DIVISIONb4236という文字は無視される。

COUNTERはFIXED BINARY(21) と宣言されており，NAMEはCHARACTER(15) と宣言されているとしよう。キーは，次のように指定することができる。

KEY (NAME || COUNTER)

COUNTERの値は，11桁の文字ストリングに変換される。(変換の規則では，この長さの２進数値を文字に変換すると，３桁のブランクと８桁の10進数値からなる11桁のストリングになると指定されている。)変換されたストリングの右端の８文字の値は，区域の指定とみなされる。したがって，キーの長さの指定がKEYLENGTH(15)のときは，NAMEの値が，比較の指定とみなされる。

キーの変換の軽減： 通常，原始キーを指定している入出力ステートメントの各々に対して，コンパイラーは，キーを固定２進数に変換するための変換ルーチンを組み入れる。次のような特別の場合には，変換ルーチンは不要である。

1. REGIONAL(1)： 原始キーが，固定２進数の要素変数または精度(p，0) をもつ定数(ただし，13≦p≦23)であるとき。

2. REGIONAL(3)： 原始キーが，次のような形式をしているとき。

   (文字ストリング式 || r)

   ただし，rは，固定２進数の要素変数または精度(p，0)をもつ定数(13≦p≦23)。

データ・セットの作成

REGIONAL(3)データ・セットは，順次的かまたは直接アクセスのどちらかによって作成することができる。

どちらの場合も，そのデータ・セットに結びつけられたファイルが開かれるときに，データ・セットは，各トラック上で使用可能なスペースの量を指定する容量レコードをつけて初期設定される。

データ・セットを作成するのにSEQUENTIAL OUTPUTファイルを使用するときは，区域番号の昇順にレコードを提示しなければならないが，連続するいくつかのレコードに対して同じ区域番号を指定してもよい。この順序にエラーがあると，KEY条件が起こる。同じ区域番号が指定されたレコードでトラックがいっぱいになると，区域番号は，自動的に１増やされる。同じ区域番号をもつレコードをさらにもっと追加しようとすると，KEY条件が起こる(順序エラー)。

緩衝域をもつファイルを用いてデータ・セットを作成しているときに，ファイルが閉じられる前に出された最後のWRITEまたはLOCATE ステートメントが，データ・セットの限界を越えてレコードを出力しようとすると，CLOSEステートメントによってエラー条件が起されることもある。

データ・セットを作成するのにDIRECT OUTPUTファイルが使用されているときは，レコードをランダムな順序で提示することができるので，キーが重複していても，何の条件も起らない。使用可能なスペースがあれば，指定されたトラック上に新しいキーが挿入される。

REGIONAL(3)データ・セットが作成されたあとは，それをアクセスするファイルは，SEQUENTIAL INPUT またはUPDATE，あるいはDIRECT INPUTまたはUPDATEとして開くことができる。既存のデータ・セット全体を削除して置き換える場合だけは，OUTPUT として開くことができる。

順次アクセス

REGIONAL(3)データ・セットをアクセスするのに用いられるSEQUENTIALファイルは，INPUTまたはUPDATE属性のどちらかをつけて開くことができる。データ転送ステートメントは，KEYオプションを含んでいてはならないが，KEYTOオプションは使用できるので，ファイルはKEYED属性をもっていてもよい。KEYTOオプションは，記録されたキーだけを，指定の変数に割り当てることを指定するものである。KEYTOオプションで指定されている文字ストリング変数の桁数が，KEYLENGTHオプションで指定されたものより多いときは，返される値(記録されたキー)は，右側にブランクをつけて拡張される。KEYLENGTHで指定されたものより桁数が少ないときは，返される値は右側で切り捨てられる。

順次アクセスは，相対トラックの昇順に行なわれる。レコードは，この順に取り出されるので，必ずしもデータ・セットに入れられた順にはならない。記録されたキーは，順次アクセスの順序には関係しない。

REGIONAL(3)データ・セットをアクセスするSEQUENTIAL UPDATEファイルに対するREADとREWRITEの両ステートメントの関係についての規則は，CONSEQUTIVEデータ・セットの場合と同じである(前述の通り)。

直接アクセス

REGIONAL(3)データ・セットを処理するのに用いられるDIRECTファイルは，INPUTまたはUPDATE属性のどちらかをつけて開くことができる。すべてのデータ転送ステートメントは，原始キーをもっていなければならない。(DIRECT属性は，KEYED属性を暗示しているため)。

1. 取出し： 指定の，記録されたキーをもつレコードが発見されない場合は，KEY条件が起こる。

2. 追加： WRITEステートメントは，指定されたトラック上にすでにあるレコードの後に，スペースがあれば，新しいレコードを挿入する。

3. 置換え： REWRITEステートメントの中で原始キーによって指定されたレコードが存在しない場合は，KEY条件が起こる。

注： 記録されたキーが重複しているいくつかのレコードが1つのトラックに含まれている場合は，重複したキーが指定されたときに得られるレコードは，わからない。

装置に結びつけられるファイル(IBM 3525カード穿孔装置)

IBM 3525カード穿孔装置は，カードの読取や印刷もできる80桁カード穿孔装置で，IBMシステム／370の場合に使用可能である。この装置用のCTLASAおよびCTL360制御文字が，図12.5および図12.6にそれぞれリストしてある。

この装置は，複数操作の機能をもっているので，装置に2つまたは3つのファイルを結びつけることによって，装置を1回通すだけの間に2つ以上の操作(読取，穿孔および印刷)を同じカードに対して行なうことができる，という利点がある。この装置，およびIBM 3505カード読取装置の使用法についての詳細が，このコンパイラーのプログラマーの手引書に述べられている。しかし，プログラムを書くときに，特定の制約について考慮する必要がある。このような制約とは，以下のようなものである。

1. 装置に結びつけられたファイルは，RECORD属性をもっていなければならないし，また，すべてがBUFFEREDか，すべてがUNBUFFEREDのどちらかでなければならない。

2. レコードは，F－形式で80桁の長さ(印刷または穿孔に結びつけられたファイルの場合は，制御バイトとしてさらに1バイト)のものでなければならない。

3. ファイルに結びつけられた読取装置または穿孔装置が開かれるとき，BUFFERSオプションの値が1にセットされる。

4. 装置に結びつけられるいくつかのファイルは，どのような順番で開いてもよいが，そのいずれかへの(またはそこからの)転送が行なわれる前にすべてのファイルを開いておかなければならない。

5. 結びつけられるファイルに応じて，各カードに対して行なわれる該当の操作は，必ず，読取－穿孔－印刷の順になっていなければならない。この順序の規則が守られていないと，ERROR条件が起こる。印刷操作だけは省略することができる。

6. 行の位置を決めるための制御文字を使用する印刷関連のファイルは，カードの送込みをしないようにしなければならない。そのカードに対する最大の行数(2または25)を越えて印刷する命令が出されたとき，あるいは新しいレコードを意味する制御文字が使用されているときに，このようなことが起こる。たとえば，'1'という制御文字は，次のカードの最初の行に印刷することを指定するものである。

7. 装置に結びつけられるいくつかのファイルは，通常，どのような順序で閉じてもよいが，それらのファイルが1つでも閉じられたあとは，転送をすることはできない。したがって，BUFFERED OUTPUTファイルに対してLOCATEステートメントを使用するときは，注意しなければならない。LOCATEステートメントからの出力は，そのファイルに対する次のLOCATE，WRITE，またはCLOSEステートメントが与えられるまで，実際には行なわれない。印刷関連のファイルと穿孔関連のファイルの両方にLOCATEステートメントを使用しているときは，印刷ファイルより前に穿孔ファイルを指定するようにして，複数閉止のCLOSEステートメントを使用しなければならない。たとえば，

```
LOCATE A FILE(PUNCHOUT);
LOCATE B FILE(PRINTOUT);
CLOSE FILE(PUNCHOUT),FILE(PRINTOUT);
```

レコード方式転送の要約

次の事項は，レコード方式転送の主な特徴を上げたものである。

1. SEQUENTIALファイルは，レコードを物理的な順に処理してゆくこと(INDEXEDまたはキー順のVSAMデータ・セット，あるいはBACKWARDSファイルの場合をのぞく)を指定している。INDEXEDまたはキー順のVSAMデータ・セットは，キーの昇順に処理される。BACKWARDSファイルは，逆の物理的な順序で処理される。

2. DIRECT ファイルは，レコードをランダムな順序で処理できることを指定している。所要のレコードは，キーによって識別する。

3. SEQUENTIALファイルによってアクセス，作成または修正されるデータ・セット内のレコードは，記録されたキーをもっていても，いなくてもよい。もっているときは，記録されたキーは，KEYTO および KEYFROM オプションによってそれぞれ，データ・セットから取り出したり，入れたりすることができる。

4. INDEXEDまたはキー順のVSAMデータ・セットは，KEYED 属性をもつSEQUENTIAL INPUTまたはSEQUENTIAL UPDATEファイルを用いて，次のようにしてアクセスすることができる。ファイルは，所要のレコードのキーを指定するREAD KEY操作によって，データ・セット内の所定のレコードに位置づけされる。

5. SEQUENTIAL UPDATEファイルのデータ・セットの既存レコードは，書き直したり，修正したりすることもできるし，無視することもできる。WRITE ステートメントを使って，DIRECT UPDATEファイルにレコードを追加することができる。UPDATE ファイル内の既存のレコードは，REWRITE ステートメントを使って置き換えることができる。DELETE ステートメントを使ってレコードを削除したあと，そのレコードが入っていたスペースを再び要求することができる。

6. REWRITE ステートメントがレコードを置き換えるのに対して，WRITE ステートメントは，レコードをデータ・セットに追加する。したがって，WRITE ステートメントは，OUTPUT ファイルでも，DIRECT UPDATE ファイルでも使用することができるが，REWRITE ステートメントは，UPDATE ファイルでしか使用することができない。さらに，DIRECTファイルの場合，REWRITE ステートメントは，置き換えられる既存のレコードを識別するためにKEYオプションを使用するのに対して，WRITE ステートメントは，KEYFROM オプションを使用する。このオプションは，データ・セットの中にレコードを書き出す位置を指定するばかりでなく，REGIONAL(1)データ・セットの場合以外は，データ・セットの中に記録される識別キーも指定する。

7. SEQUENTIAL INPUTまたはSEQUENTIAL UPDATE ファイルのレコードは，READ ステートメントの IGNORE オプションを使用することによって，飛ばしたり，無視したりすることができる。IGNORE オプションの式は，飛ばされるレコードの数を指定するものである。FILE オプションだけが指定されているREAD ステートメントは，1レコードをスキップすることを指示するものである。

レコードファイルの宣言の例

次にあげるのは，ファイル定数の宣言の例である。

ENVIRONMENT 属性は，つねに指定しなければならないし，また，MEDIUMオプションでつねに，論理装置の名前とそのデータ・セットに対して使用される物理装置のタイプを指定しなければならない。

例1：

```
DECLARE FILE#3 INPUT DIRECT
    ENVIRONMENT(U RECSIZE(328)
     REGIONAL(3)
     KEYLENGTH(8)
     MEDIUM(SYS001, 2314));
```

この宣言は，3つのファイル属性（INPUT, DIRECT, およびENVIRONMENT）だけを指定している。この他に暗示されている属性は，FILE（上記の属性のいずれからも暗示される），RECORD, KEYED（DIRECT によって暗示される），およびUNBUFFERED（これもDIRECTから暗示される）である。有効範囲は，省略時解釈によりEXTERNALである。ENVIRONMENT属性は，データ・セットがREGIONAL(3)編成のものであり，最高328バイトの長さの不定長レコードをもっていることを指定している。KEYオプションは，このファイルを参照する各々のREAD ステートメントの中で指定しなければならない。

例2：

```
DECLARE INVNTRY UPDATE BUFFERED
    ENVIRONMENT (F RECSIZE(100)
     INDEXED
     KEYLENGTH(8)
     MEDIUM(SYS001, 2314));
```

この宣言も，3つのファイル属性だけ（UPDATE, BUFFERED, およびENVIRONMENT）を指定している。暗示される属性は，FILE, RECORD, およびSEQUENTIAL（後の2つの属性は，BUFFEREDによって暗示される）である。有効範囲は，省略時解釈によりEXTERNALである。このデータ・セットは，INDEXED 編成であり，各々100バイトの固定長レコードを含んでいる。このデータ・セットには，実際には記録されたキーが含まれているが，KEYED 属性が指定されていないので，READ ステートメントの中でKEYTO オプションを指定することはできない。

```
DECLARE INVNTRY FILE SEQUENTIAL
     UNBUFFERED
    ENVIRONMENT(F RECSIZE(100)
     CONSECUTIVE
    MEDIUM(SYS001, 2314));
```

このような宣言では，INVNTRYは,別の目的のために開くことができる。たとえば，これは次のようにして開いてもよい。

```
OPEN FILE (INVNTRY) OUTPUT;
```

このファイルは，上のようにして開き，次に閉じて，さらにのちに開き直すこともできる。たとえば，

```
OPEN FILE (INVNTRY) INPUT;
```

この機能は，CONSECUTIVE編成のUNBUFFERED SEQUENTIALファイルの場合にだけ許されるもので，OPENステートメントでは，INPUTまたはOUTPUT属性しか指定することができない。

ファイル宣言[1]	有効なステートメント[2]，および記入すべきオプション	使用してもよい他のオプション
SEQUENTIAL OUTPUT BUFFERED	WRITE FILE(ファイル式) FROM(変数); LOCATE 変数 FILE(ファイル式);	 SET(ポインター変数)
SEQUENTIAL OUTPUT UNBUFFERED	WRITE FILE(ファイル式) FROM(変数);	EVENT(事象変数)
SEQUENTIAL INPUT BUFFERED	READ FILE(ファイル式) INTO(変数); READ FILE(ファイル式) SET(ポインター変数); READ FILE(ファイル式) IGNORE(式);	
SEQUENTIAL INPUT UNBUFFERED	READ FILE(ファイル式) INTO(変数); READ FILE(ファイル式) IGNORE(式);	EVENT(事象変数) EVENT(事象変数)
SEQUENTIAL UPDATE BUFFERED	READ FILE(ファイル式) INTO(変数); READ FILE(ファイル式) SET(ポインター変数); READ FILE(ファイル式) IGNORE(式); REWRITE FILE(ファイル式);	 FROM(変数)
SEQUENTIAL UPDATE UNBUFFERED	READ FILE(ファイル式) INTO(変数); READ FILE(ファイル式) IGNORE(式); REWRITE FILE(ファイル式) FROM(変数);	EVENT(事象変数) EVENT(事象変数) EVENT(事象変数)

[1] 完全なファイルの宣言には，FILE，RECORD，およびENVIRONMENTの諸属性が含められる。
[2] ステートメント： READ FILE(ファイル式)；は正しいステートメントであり，
READ FILE(ファイル式) IGNORE(1)；と同じである。

図12.10 CONSECUTIVEおよびエントリー順のVSAMデータ・セットの作成とアクセスに使用できるステートメントとオプション

ファイル宣言[1]	有効なステートメント[2]，および記入すべきオプション	使用してもよい他のオプション
SEQUENTIAL OUTPUT	WRITE FILE(ファイル式) FROM(変数) KEYFROM(式)；	
	LOCATE 変数 FILE(ファイル式) KEYFROM(式)；	SET(ポインター変数)
SEQUENTIAL INPUT	READ FILE(ファイル式) INTO(変数)；	KEY(式) または KEYTO (文字ストリング変数)
	READ FILE(ファイル式) SET(ポインター変数)；	KEY(式) または KEYTO (文字ストリング変数)
	READ FILE(ファイル式) IGNORE(式)；	
SEQUENTIAL UPDATE	READ FILE(ファイル式) INTO(変数)；	KEY(式) または KEYTO (文字ストリング変数)
	READ FILE(ファイル式) SET(ポインター変数)	KEY(式) または KEYTO (文字ストリング変数)
	READ FILE(ファイル式) IGNORE(式)；	
	REWRITE FILE(ファイル式)；	FROM(変数)
DIRECT INPUT	READ FILE(ファイル式) INTO(変数) KEY(式)；	EVENT(事象変数)
DIRECT UPDATE	READ FILE(ファイル式) INTO(変数) KEY(式)；	EVENT(事象変数)
	REWRITE FILE(ファイル式) FROM(変数) KEY(式)；	EVENT(事象変数)
	WRITE FILE(ファイル式) FROM(変数) KEYFROM(式)；	EVENT(事象変数)

[1] 完全なファイルの宣言には，FILE，RECORD，およびENVIRONMENTの諸属性が含まれる。KEY，KEYFROM，および KEYTOのいずれかが使用されているときは，KEYED属性も含まれていなければならない。

[2] ステートメント： READ FILE(ファイル式)；は，ステートメント：
READ FILE(ファイル式) IGNORE(1)；と同じである。

注： SEQUENTIALファイルの場合は，UNBUFFERED属性は無視され，BUFFEREDとみなされる。

図12.11 INDEXEDデータ・セットの作成とアクセスに使用できるステートメントとオプション

ファイル宣言[1]	有効なステートメント[2]，および記入すべきオプション	使用してもよい他のオプション
SEQUENTIAL OUTPUT BUFFERED	WRITE FILE(ファイル式) FROM(変数) KEYFROM(式)； LOCATE 変数 FILE(ファイル式) KEYFROM(式)；	 SET(ポインター変数)
SEQUENTIAL OUTPUT UNBUFFERED	WRITE FILE(ファイル式) FROM(変数) KEYFROM(式)；	EVENT(事象変数)
SEQUENTIAL INPUT BUFFERED	READ FILE(ファイル式) INTO(変数)； READ FILE(ファイル式) SET(ポインター変数)； READ FILE(ファイル式) IGNORE(式)；	KEYTO(文字ストリング変数) KEYTO(文字ストリング変数)
SEQUENTIAL INPUT UNBUFFERED	READ FILE(ファイル式) INTO(変数)； READ FILE(ファイル式) IGNORE(式)；	EVENT(事象変数) および／または KEYTO (文字ストリング変数) EVENT(事象変数)
SEQUENTIAL UPDATE BUFFERED	READ FILE(ファイル式) INTO(変数)； READ FILE(ファイル式) SET(ポインター変数)； READ FILE(ファイル式) IGNORE(式)； REWRITE FILE(ファイル式)；	KEYTO(文字ストリング変数) KEYTO(文字ストリング変数) FROM(変数)
SEQUENTIAL UPDATE UNBUFFERED	READ FILE(ファイル式) INTO(変数)； READ FILE(ファイル式) IGNORE(式)； REWRITE FILE(ファイル式) FROM(変数)；	EVENT(事象変数) および／または KEYTO (文字ストリング変数) EVENT(事象変数) EVENT(事象変数)
DIRECT OUTPUT	WRITE FILE(ファイル式) FROM(変数) KEYFROM(式)；	EVENT(事象変数)
DIRECT INPUT	READ FILE(ファイル式) INTO(変数) KEY(式)；	EVENT(事象変数)
DIRECT UPDATE	READ FILE(ファイル式) INTO(変数) KEY(式)； REWRITE FILE(ファイル式) FROM(変数) KEY(式)； WRITE FILE(ファイル式) FROM(変数) KEYFROM(式)；	EVENT(事象変数) EVENT(事象変数) EVENT(事象変数)

[1] 完全なファイルの宣言には，FILE，RECORD，およびENVIRONMENTの諸属性が含まれる。KEY，KEYFROMおよびKEYTO のいずれかが用いられているときは，KEYED属性も含まれていなければならない。

[2] ステートメント： READ FILE(ファイル式)；は，ステートメント：
READ FILE(ファイル式) IGNORE(1)；と同じである。

図12.12 REGIONALデータ・セットの作成とアクセスに使用できるステートメントとオプション

ファイル宣言[1]	有効なステートメント[2]，および記入すべきオプション	使用してもよいその他のオプション
SEQUENTIAL OUTPUT BUFFERED	WRITE FILE(ファイル式) FROM(変数) KEYFROM(式)； LOCATE 変数 FILE(ファイル式) KEYFROM(式)；	 SET(ポインター変数)
SEQUENTIAL OUTPUT UNBUFFERED	WRITE FILE(ファイル式) FROM(変数) KEYFROM(式)；	EVENT(事象変数)
SEQUENTIAL INPUT BUFFERED	READ FILE(ファイル式) INTO(変数)； READ FILE(ファイル式) SET(ポインター変数)； READ FILE(ファイル式) IGNORE(式)；	KEY(式) または KEYTO (文字ストリング変数) KEY(式) または KEYTO (文字ストリング変数)
SEQUENTIAL INPUT UNBUFFERED	READ FILE(ファイル式) INTO(変数)； READ FILE(ファイル式) IGNORE(式)；	EVENT(事象変数) および／またはKEY (式) またはKEYTO (文字ストリング変数) のいずれか EVENT(事象変数)
SEQUENTIAL UPDATE BUFFERED	READ FILE(ファイル式) INTO(変数)； READ FILE(ファイル式) SET(ポインター変数)； READ FILE(ファイル式) IGNORE(式)； REWRITE FILE(ファイル式)； DELETE FILE(ファイル式)；	KEY(式) または KEYTO (文字ストリング変数) KEY(式) または KEYTO (文字ストリング変数) FROM(変数) KEY(式)
SEQUENTIAL UPDATE UNBUFFERED	READ FILE(ファイル式) INTO(変数)； READ FILE(ファイル式) IGNORE(式)； REWRITE FILE(ファイル式) FROM(変数)； DELETE FILE(ファイル式)；	EVENT(事象変数) および／またはKEY (式) またはKEYTO (文字ストリング変数) のいずれか EVENT(事象変数) EVENT(事象変数) KEY(式) および／または EVENT(事象変数)

図12.13（2の1） キー順のVSAMデータ・セットの作成およびアクセスに使用できるステートメントとオプション

ファイル宣言[1]	有効なステートメント[2]，および記入すべきオプション	使用してもよいその他のオプション
DIRECT INPUT BUFFERED	READ FILE(ファイル式) INTO(変数) KEY(式)； READ FILE(ファイル式) SET(ポインター変数) KEY(式)；	
DIRECT INPUT UNBUFFERED	READ FILE(ファイル式) INTO(変数) KEY(式)；	EVENT(事象変数)
DIRECT UPDATE BUFFERED	READ FILE(ファイル式) INTO(変数) KEY(式)； READ FILE(ファイル式) SET(ポインター変数) KEY(式)； REWRITE FILE(ファイル式) FROM(変数) KEY(式)； DELETE FILE(ファイル式) KEY(式)； WRITE FILE (ファイル式) FROM(変数) KEYFROM(式)；	
DIRECT UPDATE UNBUFFERED	READ FILE(ファイル式) INTO(変数) KEY(式)； REWRITE FILE(ファイル式) FROM(変数) KEY(式)； DELETE FILE(ファイル式) KEY(式)； WRITE FILE(ファイル式) FROM(変数) KEYFROM(式)；	EVENT(事象変数) EVENT(事象変数) EVENT(事象変数) EVENT(事象変数)

[1] 完全なファイルの宣言には，FILE およびRECORD属性が含まれる。KEY，KEYFROM，およびKEYTOのいずれかが使用されているときは，KEYED属性も宣言に含めなければならない。

[2] ステートメント： READ FILE(ファイル式)；は，ステートメント；
READ FILE(ファイル式) IGNORE(1)；と同じである。

図12.13(2の2) キー順のVSAMデータ・セットの作成とアクセスに使用できるステートメントとオプション

第１３章：　編集およびストリングの取扱い

PL／Iでは，算術演算子，比較演算子，およびビット・ストリング演算子によって行なわれるデータ操作は，ビット取扱いおよび編集に関する様々な機能によって，展開される。これらの機能は，データ属性，ステートメントのオプション，組込み関数，および擬似変数によって指定する。

このあと，これらの各機能について，例題をあげながら概説する。

割当てによる編集

割当てステートメントによって実行される編集の最も基本的な形式には，割当て記号の右辺の値のデータ・タイプを，受け入れ側の変数の属性に合うように変換することが含まれている。割り当てられる値は，受け入れ側のフィールドの属性に合わせられるので，割り当てられる値の精度または長さは，変更されることもある。このような変更には，変換される項目に数字または文字を追加すること，および変換される項目から数字または文字を削除することも含めることができる。データ変換の規則については，第４章「式とデータ変換」およびＦ節「データ変換と式の値の計算」のところに述べられている。

ストリング・データの長さの変更

ストリング変数に値が割り当てられるときは，必要に応じて，受け入れ側ストリングと同じストリング・タイプ（文字またはビット）に値が変換される。必要があれば，受け入れ側のストリングについて宣言された長さに合わせて，切り捨てられたり，また受け入れ側のストリングが固定長のときは，右側に拡張されたりする。たとえば，SUBJECT が，１０桁の固定長文字ストリングであることを示すCHARACTER（１０）という属性をもっているとして，次のステートメントについて考えてみよう。

```
SUBJECT = 'TRANSFORMATIONS';
```

右辺のストリングの長さは，１５桁である。したがって，それがSUBJECT に割り当てられるときは，ストリングの右端から５桁が切り捨てられる。これは，次のものを実行するのと同じである。

```
SUBJECT = 'TRANSFORMA';
```

割り当てられるストリングが，受け入れ側のストリング変数に対して宣言された長さよりも短いときは，割り当てられるストリングは，右側にブランク（文字ストリング変数のとき），またはゼロ（ビット・ストリング変数のとき）が拡充される。SUBJECTがやはりCHARACTER（１０）という属性をもっているとすれば，次の２つのステートメントは，SUBJECT に対して同じ値を割り当てることになる。

```
SUBJECT = 'PHYSICS';
SUBJECT = 'PHYSICSbbb';
```

bという文字は，ブランク文字を表わしている。

CODEは，BIT（１０）という属性をもつビット・ストリング変数であるとすれば，次の２つのステートメントは，CODEに対して同じ値を割り当てることになる。

```
CODE = '110011'B;
CODE = '1100110000'B;
```

しかし，SUBJECTがCHARACTER（１０）という値をもっているとすれば，次の２つのステートメントは，SUBJECT に対して同じ値を割り当てることにはならない。

```
SUBJECT = '110011'B;
SUBJECT = '1100110000'B;
```

最初のステートメントが実行されると，右辺のビット・ストリング定数が最初に文字ストリングに変換され，次に，右側に，ブランク文字（ゼロ・ビットではなく）が拡充される。このステートメントは，次のものと同じである。

```
SUBJECT = '110011bbbb';
```

上の２つのうち２番目のステートメントは，ビット・ストリングから文字ストリング・タイプへの変換を必要とするだけで，次のものに等しい。

```
SUBJECT = '1100110000';
```

しかし，VARYING 属性をもつストリング変数に割り当てられるときは，ストリング値は，ブランク文字またはゼロ・ビットによって拡張されない，かわりに，受け入れ側のストリング変数の長さ指定が，割り当てられる各々のストリングの長さを記述するようにうまく調整される。しかし，割り当てられるストリングの長さが，長さの変わるストリング変数に対して宣言された最大の長さを越えているときは，切捨てが行なわれる。

割当ての別の形式

割当てステートメントのほかに，PL/I には，変数に値を割り当てるための別の方法がいくつかある。このうち，入出力ステートメントに関係する方法は2つある。1つは，実際に入出力操作が行なわれるもので，もう1つは，全く内部的なデータ移動の方法である。

入出力操作

割当てステートメントは，コンピューター内部での記憶位置間でのデータ転送に関するものであるが，入出力操作は，コンピューターの内部記憶装置と外部記憶装置の間で転送が行なわれるような割当ての，関連形式とみなすこともできる。

しかし，レコード方式の操作では，転送されるときに論理レコードの中にある項目のデータ変換は行なわれない。レコードの編集が必要ならば，レコードを書き出す前または読み込んだあとで，内部記憶域の中で行なわなければならない。

それに対して，ストリーム方式の操作では，データ項目を読んだり，書いたりするときに使用できる編集機能がいくつも用意されている。これらの編集機能は，割当てステートメントで行なわれる機能とほぼ同じであるが，どのようなデータ変換にもつねに文字タイプが関係する点（入力のときは文字タイプからの変換，出力のときは文字タイプへの変換）は，例外である。

GETおよびPUTステートメントのSTRING オプション

GETおよびPUTステートメントの中でSTRING オプションを使うことによって，内部記憶装置と外部記憶装置の間ではなく，内部記憶位置同志の間でデータを転送するために，これらのステートメントを使用することができる。GETとPUTのどちらのステートメントの場合も，FILE（ファイル式）によって指定されるFILEオプションは，次の形式で示すように，STRINGオプションに置き換えられる。

GET STRING（文字ストリング名）データ指定；

PUT STRING（文字ストリング名）データ指定；

GETステートメントは，データ・リストの中の変数に割り当てられるデータ項目が，指定された文字ストリングから得られることを指定している。PUTステートメントは，データ・リストのデータ項目を，指定された文字ストリング変数に割り当てることを指定している。“データ指定”は，入力と出力のところで述べたものと同じである。一般に，これは次のような形式をしている。

```
DATA 〔（データ・リスト）〕
〔LIST〕（データ・リスト）
EDIT {（データ・リスト）（形式リスト）} ・・・
```

STRING オプションは，ストリーム方式転送の3つのモードの各々で使用することができるが，編集指示転送（このモードでは，入力の流れは連続する文字のストリングとみなされる）の場合に，最も効果がある。リスト指示およびデータ指示のGETステートメントの場合，文字ストリングの中の個々の項目は，カンマまたはブランクによって区切らなければならない。データ指示のGETステートメントでは，転送を止めるセミコロンをストリングにも含めなければならないし，また各データ項目を，割当てステートメントの形式の中に記入しなければならない。編集指示転送は，形式リストを用いて編集機能を行なう。COLUMN 制御形式項目は，STRING オプションといっしょに使用できないことに注意していただきたい。

STRING オプションをもつGET DATAステートメントでは，NAME条件は起らない。かわりに，FILE オプションをもつGET DATAステートメントに対してNAME条件を引き起すような状況のときに，ERROR 条件が起こる。

STRINGオプションを用いると，1つのステートメントでデータの集合と分散の操作を行なうことができるようになり，レコード方式のステートメントによって転送される文字ストリングをストリーム方式によって処理することが可能になる。次のステートメントについて考えてみよう。

```
PUT STRING (RECORD) EDIT
  (NAME, PAY#, HOURS*RATE)
  (A(12), A(7), P'$999V.99');
```

このステートメントは，NAMEの文字ストリング値を，RECORDという名前のストリングの最初の（左端の）12桁に割り当てること，およびPAY# の文字ストリング値を，RECORD の次の7桁に割り当てること，を指定している。次に，HOURS の値とRATE の値が乗算され，その積が，ピクチャー指定にしたがって編集されて次の7桁に入れられる。

いくつかの異なる形式のレコード（それぞれ，自分自身の中にあ

るデータ項目の値によって形式の表示をしている）を読まなければならないことがよくある。STRINGオプションを使うと，このようなレコードを処理するのが容易になる。たとえば，

```
READ FILE (INPUTR) INTO (TEMP);
GET STRING (TEMP) EDIT (CODE) (F(1));
IF CODE¬= 1 THEN GO TO OTHER_TYPE;
GET STRING (TEMP) EDIT (X,Y,Z)
    (X(1), 3F(10,4));
```

READステートメントは，入力ファイルINPUTRからレコードを読む。最初のGETステートメントは，レコードの最初のバイトからコードを取り出し，それをCODEに割り当てるためにSTRINGオプションを使用している。このコードは，レコードの形式を判定するためにテストされる。コードが1のときは，2番目のGETステートメントは，STRINGオプションを用いて，レコード内の項目をX，Y，およびZに割り当てる。2番目のGETステートメントは，ストリングTEMPの中の最初の文字を無視する（形式リストの中のX(1)形式項目）よう指定していることに注意していただきたい。STRINGオプションをもつ各々のGETステートメントはつねに，ストリングの最初の文字から走査を始めることを指定している。したがって，2番目のGETステートメントで無視される文字は，最初のGETステートメントによってCODEに割り当てられるものと同じ文字である。

同じようにして，STRINGオプションをもつPUTステートメントを用いて，内部記憶装置の中にレコードを作成することができる。次の例の中で，ファイルOUTPRTは，最終的には印刷されるものとしよう。

```
PUT STRING (RECORD) EDIT
    (NAME,PAY#, HOURS*RATE)
    (X(1), A(12), X(10), A(7),X(10),
     P'$999V.99');
WRITE FILE (OUTPRT) FROM (RECORD);
```

このPUTステートメントは，文字ストリング変数に割り当てられる最初の文字は1個のブランク，すなわち印刷の前に1スペースとることを指定するキャリッジ制御文字であることを，X(1)スペース形式項目を使って指定している。さらにその後で，変数NAMEおよびPAY#の値とHOURS*RATEという式の値を割り当てることを指定している。形式リストでは，NAMEとPAY#の間，およびPAY#と式の値の間にブランクを挿入することが指定されている。WRITEステートメントは，レコードをファイルOUTPRTに書き出すためにレコード転送を用いることを指定している。

ピクチャー指定

ピクチャー指定は，PL/Iで使用できる編集機能を発展させたもので，これを使うことによって，プログラマーは，データ形式を大巾に制御することができる。ピクチャー指定は，一連の文字コードを引用符で囲んだものから構成されるが，この引用符もPICTURE属性の一部分，またはP（ピクチャー）形式項目の一部分である。

```
DECLARE PRICE PICTURE'$Z9V99';

PUT FILE(SYSPRINT)EDIT('PART NUMBER',
    PART#)(A(12),P'AAA99X');
```

ピクチャー指定には，次の2つのタイプがある。

- 数字指定
- 文字ストリング・ピクチャー指定

PICTURE属性の数字指定は，データ項目は数量を表わしているが，文字ストリングとして貯えられること，およびその数値がストリングの中でどのように表現されるかということを表示するものである。P形式項目の数字指定は，数値が，外部媒体上で文字ストリングとしてどのように表現されているか（あるいはこれから表現されるか）を表示するものである。

文字ストリング・ピクチャー指定は，固定長の文字ストリングを記述するためのもう1つの方法であり，ストリング内の任意の位置に，オペレーティング・システムで使用できる完全な文字セットのうちある特定のサブセットからの文字だけを入れることができることを表示する追加機能をもっている。

2つのタイプのピクチャー指定の概念については，以下に別々に説明するが，個々のピクチャー文字の詳細とその用法の例は，D節「ピクチャー指定文字」のところでとりあげてある。ここでは，AまたはXというピクチャー文字があれば，そのピクチャー指定は，文字ストリング・ピクチャー指定として定義されているのであり，それ以外は数字指定である，ということに注意するだけでよい。

数字指定

数字指定は，関連するデータ項目は数値であるが，文字ストリングとして貯えられる（あるいは，外部媒体上で表現されている）ことを指定するものである。また，これは，文字ストリングがとる形式や，ストリングの中でどのように数値が表現されるかということも指定する。たとえば，

```
DCL PRICE PICTURE'$Z9V99';
```

これは，PRICEが，長さ5の文字ストリングによって表わされることを指定している。最初の文字はつねに$，2番目の文字はブランクまたは非ゼロの数字，3，4，5番目の文字は数字である。

この数値は，数値を表わすことのできる4文字によって表示され，Vによって示される位置に小数点があるとみなされる。したがって，PRICEは，FIXED DECIMAL(4,2)とみなされ，つねに正である。13.25は，'$1325'と表わされ，.95は'$b095'と表わされる。

数字指定には，次のような2つの主な用法がある。

- 入出力操作に関連するデータ項目に対して(これらの項目は，数値データを入れることのできるところであれば，プログラム内のどこででも使用することができるが)。しかし，ピクチャーの与えられたデータに対する数値操作の多くは，コード化された数値データに対して同じ操作を行なった場合にくらべて，著しく効率が悪くなる。

- 2番目の用法は，ピクチャーの与えられた1つのデータ項目が実際上2つの値をもつことを止める働きをする。その項目が数値の文脈の中で用いられているときは，数値は，ピクチャー指定によって定義された変換処理によって，文字ストリングから取り出されたり，文字ストリングに入れられたりする。その項目が，文字ストリング式が必要な文脈の中で原データとして使用されているときは，その値を表わす実際の文字ストリングが用いられる。たとえば，

```
DCL COUNT PICTURE'999' INITIAL(0),
    STRING CHAR (3) ;
COUNT = COUNT +1;
STRING = COUNT;
```

COUNT の最初の表現は，'000' である。最初の割当てステートメントでは，これは，FIXED DECIMAL (3,0) に変換され，加算が行なわれて，その結果が，ピクチャー形式'001'に変換し直される。2番目の割当てステートメントでは，ストリングの値が'001' にセットされる。

定義の中で文字ストリング値を使用してもよいことにとくに注意していただきたい。数字データ項目は，文字ストリングに重ねて定義してもよく，またその逆も可能である。

数字指定の中のピクチャー文字'9'

ピクチャー文字9は，数字ピクチャー指定の最も簡単な形式である。n個のピクチャー文字9のストリングは，その項目が，n桁の固定長文字ストリングによって表わされること，またその各桁は数字(0から9)であることを指定している。この数値は，無符号の10進数(つまり，FIXED DECIMAL(n,0))としての数字の値である。たとえば，

```
DCL DIGIT PICTURE'9'
    COUNT PICTURE'999',
    XYZ PICTURE '(10)9';
```

XYZの宣言は，ピクチャー文字9を10回書くことにかわる別の方法を示すものである。

例:

```
DCL 1 CARD_IMAGE,
      2 DATA CHAR(72),
      2 IDENTIFICATION CHAR(3),
      2 SEQUENCE PIC'99999';
   •
   •
   •
SEQUENCE = SEQUENCE + 1;
WRITE FILE(OUTPUT) FROM(CARD_IMAGE);
```

文字ストリング・ピクチャーの中の9の定義は，対応する文字がブランクまたは数字でよいという点で異なっている。

ピクチャー文字 Zおよび*

数の中の先行ゼロをブランクで置き換えたい場合がある。ピクチャーの中では，Zピクチャー指定文字を使うことによって，このようなことが可能になる。Zと9だけが含まれるピクチャー指定は，1つまたは数個のZの後に，任意選択で1つまたは数個の9をつづけた形になる。数値データの表現は，その中に入れられる数字が普通ならばゼロになるはずであるが，その左側にあるすべての桁もゼロであるようなときは，文字ストリングのこの桁はブランクになるという点をのぞけば，'9'ピクチャー指定の場合と同じである。たとえば，

```
DCL PAGE_NUMBER PICTURE'ZZ9';
```

197という値は，'197'になり，69は'b69'，5は'bb5'，ゼロは'bb0'になる。全体がZのピクチャー指定の場合は，ゼロの値は，全桁がブランクのストリングになる。

星印のピクチャー指定文字は，Zという文字と同様の効果をもっているが，ブランクのかわりに*がストリングの中に入れられる点が異なっている。たとえば，これは，小切手を印刷するときに，フィールド内にブランクのスペースを残したくない場合などに使用することができる。たとえば，

```
DCL CREDIT PICTURE '$**9.99';
```

($および(.)という文字については，次に説明する。)95という値は'$**0.95'になり，12350という値は，'$123.50'になる。

ピクチャー文字V

ピクチャー指定文字Vは，文字ストリング内の仮想上の小数点の位置を示すものである。たとえば，

```
DCL VALUE PICTURE 'Z9V999';
```

ストリング'12345'は，数値12.345を表わしている。ピクチャー指定の中にあるVという文字は，文字ストリング表現の中にある文字の位置を指定するものではないことに注意していただきたい。とくに，データ項目への割当ての際に，小数点は，文字ストリングの中に含まれない。

ピクチャー文字B　.　,　/　の挿入

小数点ピクチャー文字(·)を，数値ピクチャー指定の中に入れることができる。これは単に，文字表現の値の中に，小数点が含まれることを指示するものである。したがって，小数点は，その文字ストリング値の一部分である。小数点ピクチャー文字があっても，これは変数の算術値の一部分ではないため，割付けの際に小数点の位置合せは行なわれない。文字Vだけが，小数点の位置合せを行なう。たとえば，

```
DECLARE SUM PICTURE '999V.99';
```

SUMは，3番目を4番目の数字の間に仮想上の小数点がある5桁の数字からなる数を表わす数字変数である。小数点挿入文字によって指定される実際の小数点は，算術値の一部分ではなく，その文字ストリング値の一部である。（小数点ピクチャー文字は，文字Vの左か右のどちらかに置くことができる。ある種の文脈では，左にするか右にするかで意味が違うこともあるが，多くの場合は，V.にする方がよい。D節「ピクチャー指定文字」を参照のこと。）次にあげる2つのステートメントは，同じ文字ストリングをSUMに割り当てるものである。

```
SUM = 123;

SUM = 123.00;
```

最初のステートメントの場合は，123という数字の右側に2桁のゼロが付け加えられる。

次の宣言の影響について注意していただきたい。

```
DECLARE RATE PICTURE '9V99.99';
```

RATEは，次のように使用するものとしよう。

```
RATE = 7.62;
```

このステートメントが実行されるとき，小数点の位置合せは，文字Vのところで行なわれるのであって，RATEに対するピクチャー指定の中の小数点ピクチャー文字のところではない。RATEが印刷されるとすれば，'762.00'と印刷されるが，その算術値は7.6200である。

文字Vは，ピクチャー指定の中に1つしか書くことができないが，小数点ピクチャー文字は，2回以上書くことができる。したがって，デューイの数表記法やヨーロッパ諸国の数表記によく用いられるような形式で，数字データ項目の中の数字グループを，小数点で区切ることも可能になる。

小数点ピクチャー文字は，数字データ項目の文字ストリング値に小数点を挿入するので，挿入文字と呼ばれる。PL/Iでは，このほかに3つの挿入文字（カンマ(,)，斜線(/)，およびブランク(B)）が用意されている。次のステートメントについて考えてみよう。

```
DECLARE RESULT PICTURE '9.999.999,V99';
RESULT = 1234567;
```

RESULTの文字ストリング値は，'1.234.567,00'になる。小数点の位置合せは，文字Vで指定されているように，右から2桁目の前で行なわれていることに注意していただきたい。RESULTがコード化算術フィールドに割り当てられるときは，算術形式に変換されるデータの値は，1234567.00になる。

Bという文字は，文字ストリングの対応する桁にブランクがつねに挿入される（＊ゼロ消去文字のストリングの中にあっても）という点で，他の3つの挿入文字とは違っている。

ピクチャー文字 $

ピクチャー文字$は，数字データ項目の指定の位置に通貨記号$を出すことを制御するものである。たとえば，次のステートメントで示すように，数字項目の左側にドル記号をつけることができる。

```
DECLARE PRICE PICTURE '$99V.99';

PRICE = 12.45;
```

PRICEの文字ストリング値は，文字ストリング定数'$12.45'と同じである。しかし，その算術値は，12.45である。

数字指定の中の符号の指定

符号付の情報を，数字データ項目に入れるための方法がいくつかある。この方法のうち最も簡単なものは，S文字指定である。これは，値が正またはゼロのときは'＋'，値が負のときは'－'を入れる，文字ストリング表現内の桁を指定するものである。この文字は，数字桁全体の右側，または左側のどちらかに置かなければならない。たとえば，

```
DCL ROOT PICTURE 'S999';
```

50は'+050'，ゼロは'+000'，－243は'－243'になる。同じように，'＋'ピクチャー文字は，値が正またはゼロのときは'＋'，負のときはブランクを，対応する文字桁に入れることを指定するものである。'－'ピクチャー文字は，値が正またはゼロのときはブランク，負のときは'－'を，対応する文字桁に入れることを指定するものである。

上重ね符号指定文字　TIR

符号付の値を表わすもう1つの方法がある。この方法は，上重ね穿孔符号指定によるもので，ストリングの中に追加の桁を必要としない。この表現法は，数字が含まれている桁の上部（通常は，フィールドの最後の桁）に12穿孔（＋を表わす）または11穿孔（－を表わす）を重ねることによって，穿孔カードに入っている数値データの中の符号を表わす習慣からきている，その結果できるカード・コードは，多くの場合，英字と同じである（たとえば，1から9の上に12穿孔を重ねるとAからI，1から9の上に11穿孔を重ねるとJからRになる），12－0および11－0は，PL/Iの文字セットにはないが，オペレーティング・システムによって受け入れられる文字のセットの中にはある。

ピクチャー指定文字Tは，数字と符号を上で述べたようにして入れる（つまり，正の場合は12穿孔を0から9の上に重ね，負の場合は11穿孔を0から9の上に重ねる）文字ストリング表現の中の文字を指定するものである。この文字は，ピクチャー指定文字'9'を入れることのできるところであれば，どの桁にあってもよい。たとえば，

```
DCL CREDIT PICTURE 'ZZV9T';
```

CREDIT の文字ストリング表現は，4桁である。＋21.05 は'210E'になり，－0.07は'bb0P'になる。

ピクチャー指定文字Iは，値が正またはゼロならば12穿孔を数字に重ねるが，値が負ならば数字だけの表現を入れる文字桁を指定するものである。

ピクチャー指定文字Rは，値が負ならば11穿孔を数字の上に重ねるが，値が正ならば数字だけを入れる文字桁を指定するものである。たとえば，

```
GET EDIT (X) (P'R99');
```

このステートメントは，入力ストリームの次の3桁に'132'があるとXを（＋）132にセットし，－132があると'J32'にセットする。

その他の数字機能

上記のピクチャー指定文字の用法についての詳細は，浮動符号および浮動通貨記号，浮動小数点数値に対するピクチャー指定文字の詳細と合わせて，D節「ピクチャー指定文字」のところに記載されている。

数字指定文字をすべて列挙すると，9，V，Z，*，Y，(·)，(,)，B，S，＋，－，$，CR，DB，T，I，R，K，E，Fであるが，このうちK，V，F以外は，文字ストリング表現の中に1文字が実際にあることを指定するものである。

文字ストリング・ピクチャー指定

文字ストリング・ピクチャー指定は，固定長文字ストリングを記述するためのもう1つの方法であり，使用可能な完全な文字セットの特定のサブセットからの文字だけが，そのストリングの任意の桁に入り得ることを示す機能ももっている。

文字ストリング・ピクチャー指定は，ピクチャー指定文字AまたはXを繰返し使用することによって認識される。文字ストリング・ピクチャー指定の中で使用できる文字は，X，A，および9だけである。これらは各々，文字ストリングの中で，次のものを入れることのできる1桁の位置の存在を指定している。

X　その導入システムで認識可能な文字（8ビットのバイトで表わされる256種のビットの組合せのすべて）。

A　すべての英字，#，@，$，またはブランク。

9　数字またはブランク。数字指定のピクチャー指定文字9との違いに注意。

文字ストリング値が，ピクチャーの与えられた文字ストリング・データ項目に割り当てられたり，転送されたりするときは，対応するピクチャー指定文字での指定にしたがって，各桁の特定の文字が妥当性検査を受け，無効な文字があれば，CONVERSION 条件が起こされる。たとえば，

```
DECLARE PART# PICTURE 'AAA99X';
```

次の値は，PART# に割り当てても有効である。

```
'ABC12M'
'bbb09/'
'XYZb13'
```

次の値は，正しくない（正しくない文字には,下線がつけてある）。

```
'AB123M'
'ABC1/2'
'Mb#A5;'
```

ビット・ストリングの取扱い

以下の例は，ビット・ストリングの取扱いで使用できるPL/Iの機能のいくつかを示すものである。

```
DECLARE 1 PERSONNEL_RECORD,
          2 NAME,
            3 LAST CHARACTER(15),
            3 FIRST CHARACTER(10),
            3 MIDDLE CHARACTER(1),
          2 CODE_STRING,
            3 MALE BIT(1),
            3 SECRETARIAL BIT(1),
            3 AGE,
              4 (UNDER_20,
                 TWENTY_TO_30,
                 OVER_30) BIT(1),
            3 HEIGHT,
              4 (OVER_6,
                 FIVE_SIX_TO_6,
                 UNDER_5_6) BIT(1),
            3 WEIGHT,
              4 (OVER_180,
                 ONE_TEN_TO_180,
                 UNDER_110) BIT(1),
            3 EYES,
              4 (BLUE,
                 BROWN,
                 HAZEL,
                 GREY,
                 OTHER) BIT(1),
            3 HAIR,
              4 (BROWN,
                 BLACK,
                 BLOND,
                 RED,
                 GREY,
                 BALD) BIT(1),
            3 EDUCATION,
              4 (COLLEGE,
                 HIGH_SCHOOL,
                 GRAMMAR_SCHOOL) BIT(1);
```

この構造には，文字ストリングの小構造であるNAMEと，ビット・ストリングの小構造であるCODE_STRINGが含まれている。省略時解釈によって，PERSONNEL_RECORDの要素は，UNALIGNED 属性をもっている。したがって，CODE_STRING は，1バイトに8つの要素をもつ形で，すなわち長さ25のビット・ストリングと同じようにしてマップされる。

このストリングの最初の2ビットはそれぞれ，2つの択一項目（MALEか¬MALE，および SECRETARIALか，¬SECRETARIAL）を表わしている。このほかのカテゴリー（レベル3）は，それぞれいくつかの択一項目のリストになっている。（レベル番号4とBIT(1)の属性は，各々のカテゴリーに対する因数になっている。）

PERSONNEL_RECORD に対して，次のようなプログラム部分を用いることができる。

```
INREC: READ FILE(PERSON)
            INTO (PERSONNEL_RECORD);
       IF (¬MALE & SECRETARIAL
           & UNDER_20
           & UNDER_5_6
           & UNDER_110
           & BLUE
           & (HAIR.BROWN|BLOND)
           & HIGH_SCHOOL)
           | (MALE & ¬SECRETARIAL
           & OVER_30
           & OVER_6
           & OVER_180
           & EYES.GREY
           & BALD
           & COLLEGE)
          THEN PUT LIST (NAME);
          GO TO INREC;
```

次にあげるのは，同じ情報取出し操作を行なうための別のプログラムの方法である。

```
      DECLARE PERS_STRING BIT(25)DEFINED
            CODE_STRING;
      IF PERS_STRING
            = '0110000100110000100000010'B
            THEN GO TO OUTP;
      IF PERS_STRING
            = '0110000100110000001000010'B
            THEN GO TO OUTP;
      IF PERS_STRING
            = '1000110010000010000001100'B
            THEN GO TO OUTP;
      GO TO INREC;
OUTP: PUT LIST (NAME);
      GO TO INREC;
```

ここでは，ビット・ストリングPERS_STRINGは，小構造CODE_STRINGの上に定義されている。探される情報の値を表わすために，ビット・ストリング定数が作られる。次に，このビット・ストリングは，各ビット・ストリング定数と順々に比較される。最初の定数と2番目の定数は，最初のものが茶色の髪，2番目のものがブロンドの髪のテストをするためのものである点以外は，同じであることに注意していただきたい。これらの2つの変形が，最初の例では，（HAIR. BROWN | BLOND）という形で指定されている。

PERSONNEL_RECORD をテストするための2番目の方法は，構造がALIGNED であったときは，使用できないことに注意していただきたい（オーバーレーの定義をするための基底識別名は，UNALIGNED でなければならない）。最初の方法を用いるときは，ALIGNED であった方が効率がよい。

このテストは，一連のIFステートメント（ネストされていてもよい）によって行なうこともできる。この場合は，1個のIFステートメントによって，それぞれ1つのビットがテストされる。

ストリング組込み関数

PL/Iには，数多くの組込み関数がある。そのうちのいくつかは，擬似変数としても使用することができるので，この言語のストリング取扱い機能のさらに強力な道具となっている。次に，これらの機能について概説する（さらに詳細については，G節「組込み関数と擬似変数」のところで説明する）。

BIT組込み関数は，データ項目をビット・ストリングに変換することを指定するものである。組込み関数を使用すると，プログラマーは，データ変換の標準的な規則から得られる長さを打ち消して，変換されるストリングの長さを指定することができる。

CHAR 組込み関数は，プログラマーが指定することのできる長さをもつ文字に変換される点を除いて，BIT組込み関数と全く同じである。

SUBSTR 組込み関数は，受け入れフィールドの中で擬似変数として使うこともできるもので，これを用いると，指定されたストリング値の中から指定のサブストリングを抜き出す（擬似変数の場合は割り当てる）ことができる。

INDEX 組込み関数を使用すると，指定されたサブストリング（1桁の文字またはビット）がストリング（文字ストリングまたはビット・ストリングのどちらか）の中に最初に出てくる個所を探すことができる。返される値は，サブストリングの最初の文字またはビットの位置を，ストリングの先頭から相対的に示したものである。値は，2進整数で表現される。指定のストリングの中にそのサブストリングがないときは，返される値はゼロになる。

LENGTH 組込み関数は，文字ストリングまたはビット・ストリングの現在の長さを得るためのものである。これは，VARYING属性をもつストリングの場合にとくに有効である。

HIGH 組込み関数は，照合順序の中で一番高位の文字の繰返しからなる，指定の長さをもつストリングを得るためのものである。システム／360では，この文字は16進数のFFである。

LOW組込み関数は，照合順序の中で最も低位にある文字の繰返しからなる，指定された長さのストリングを得るためのものである。システム／360およびシステム／370では，この文字は16進数の00である。

REPEAT 組込み関数は，指定されたサブストリングを繰り返して，ストリングを作るものである。これは，ストリングのパターンを作るのに用いられる。

STRING 組込み関数は，擬似変数としても使用できるもので，集合体の変数の中のすべての要素を連結して，1つのストリング要素にする。

BOOL組込み関数を使用すると，指定された2つのビット・ストリングに対して，16種のブール演算のいずれかを行なうことができる。

UNSPEC 組込み関数は，擬似変数としても使用できるもので，内部コード化表現の値は，変換せずにビット・ストリングとみなすよう，指定するものである。たとえば，

```
X = ARRAY(UNSPEC('A'));
```

上のステートメントの場合，文字'A'の内部表現は，システム／360およびシステム／370では，長さが8ビットのストリングである。このビット・ストリングは，固定小数2進数の算術値に変換され，配列に対する添字として用いられる。（この添字の10進

数値は１９３である。）

TRANSLATE 組込み関数は，ストリングの中にある指定の文字要素を，指定された置換え用の文字要素と変えるものである。置き換えられる要素が入っている，ストリング内の「桁」に，「置換え」用の要素が挿入される。

この組込み関数を使用すれば，プログラマーは，与えられたストリングの中にあるすべての文字を，他の２つのストリングの中に入っている変換テーブルにしたがって，変換するという変換機能を使用することができるようになる。これらのストリングの１つは，もう一方のストリングの中にある置換え用の文字に対するキーとして働く。たとえば，

```
DECLARE (W,X,Y,Z) CHAR (3);
        X='ABC';
        Y='TAR';
        Z='CAB';
W = TRANSLATE (X,Y,Z);

/*W = 'ART' */
```

VERIFY 組込み関数は，２つのストリングを比較して，一方のストリングの中にあるビットまたは文字が，もう一方のストリング内のどこかにあるかどうかチェックするものである。１つのストリング内のすべての文字またはビットが２番目のストリングの中に見つけられたときは，'０'という値が返される。２番目のストリングに文字またはビットがない場合は，最初のストリングの中にこの文字またはビットがあった位置を表わす値が返される。

最初のストリングは，左から右に検査される。合致しなかったビットまたは文字のうち最初のものの位置の値だけが返される。

第１４章：　例外条件の取扱いとプログラム・チェックアウト

ＰＬ／Ｉプログラムが実行されるときは，数多くの例外条件がシステムによって監視されており，それらが起るとつねに，自動的に発見されるようになっている。これらの例外条件は，オーバーフローや入出力転送エラーなどのように，エラーである場合もあるし，ファイルの終りや，出力が印刷されるときのページの終りなどのように，頻繁ではないが起ることが予想されている条件である場合もある。また，プログラムをチェックする場合，識別名のリストにあるものの１つが出てきたときは，例外条件として取り扱うよう指定することによって，選択的な流れの追跡を行なうこともできる。

テストされる条件にはそれぞれ，名前が与えられており，この名前を使って，プログラマーは，例外条件の取扱いを制御する。条件名のリストは，ＰＬ／Ｉ言語の一部分である。各条件に対するキーワード名とその説明については，Ｈ節「オンー条件」を参照していただきたい。

割込み解禁の条件と処置の設定

監視されていて，それが起ると割込みを引き起すような条件は，解禁されている　という。条件が発生して割込みが起ったときにとるよう指定されている処置は，設定された処置という。

大部分の条件は，自動的にチェックされていて，それらが起こると，システムに制御が渡り，システムは，その条件に対して指定されている標準的な処置を実行する。このような条件は，省略時解釈によって解禁されており，標準的なシステム処置が設定される。

最も一般的なシステム処置は，ＥＲＲＯＲ 条件を引き起すことである。これは，エラーのタイプを個々にチェックするのではなく，各種のタイプのエラーをまとめてチェックするのに用いられる共通の条件に対するものである。ＥＲＲＯＲ 条件に対する標準的なシステム処置は，ＦＩＮＩＳＨ 条件を起して，プログラムを停止することである。

プログラマーは，ある条件を解禁するかどうか，つまり，それが起ったときに割込みを起すためのチェックをするかどうか，指定することができる。条件が禁止されていると，その条件が起っても割込みは引き起されない。

すべての入出力条件およびＥＲＲＯＲ，ＦＩＮＩＳＨ，および ＡＲＥＡ 条件はつねに解禁されており，禁止することはできない。演算関係の条件とプログラム・チェックアウト条件は，解禁しても，禁止してもよい。プログラム・チェックアウト条件とＳＩＺＥ 条件は，割込みを起したいときは，明示的に解禁しなければならない。それ以外の条件はすべて，省略時解釈によって解禁されるので，それが発生しても割込みを起したくないときは，明示的に禁止しなければならない。

条件接頭語

ある条件を解禁するか，禁止するかは，条件接頭語によって指定することができる。条件接頭語は，１つまたは２つ以上の条件名のリスト（各条件名をカンマで区切り，全体を括弧で囲む）であり，コロンを用いてステートメント（またはステートメント・ラベル）に接続する。接頭語はつねに，ステートメントおよびステートメント・ラベルの前に置く。たとえば，

```
(SIZE):L1:X=(I**N)/(M+L);
```

接頭語リストの中にある条件名は，その接頭語の有効範囲内では，対応する条件を解禁することを表示している。接頭語として用いられる条件名の前に，ブランクや連結記号で区切らずにＮＯという語をつけることによって，対応する条件が禁止されることを表示することもできる。たとえば，

```
(NOCONVERSION):Y=A||B;
```

条件接頭語の有効範囲

接頭語の有効範囲，つまり，その接頭語が適用されるプログラムの部分は，通常，その接頭語がつけられているステートメントである。そのステートメントの実行中に呼び出される関数またはサブルーチンには，適用されない。

ＩＦステートメントにつけられた条件接頭語は，ＩＦの後にある式の評価に対してだけ適用されるもので，ＴＨＥＮ またはＥＬＳＥ クローズの中のステートメントには適用されない（ＴＨＥＮ や ＥＬＳＥ 自体が接頭語をもつことは差支えないが），同様に，ＯＮ

ステートメントにつけられた接頭語は，オン・ユニットの中にあるステートメントに対しては効果をもたない。DOステートメントに対する条件接頭語は，DOステートメント自体の式の評価に対してだけ適用され，DOグループ内の他のステートメントには適用されない。

PROCEDUREステートメントおよびBEGINステートメントにつけられる条件接頭語は，特別なケース（一般によく使用されるが）である。PROCEDURE またはBEGIN ステートメントにつけられた条件接頭語は，対応するENDステートメントのところまでにあるすべてのステートメント（ENDも含めて）に適用される。この中には，そのブロックの中にネストされた他のPROCEDUREまたはBEGINステートメントも含まれる。これは，そのブロックの外側にあるプロシージャーには適用されない。

ブロックの中にあるステートメント（PROCEDURE およびBEGIN ステートメントも含めて）に接頭語をつけることによって，そのブロック内の条件の解禁または禁止の状態を定義し直すこともできる（したがって当然，ネストされたブロックの中の条件の解禁または禁止も定義し直される）。このような再定義は，接頭語がつけられるステートメントの実行にだけ適用される。ネストされたPROCEDURE またはBEGIN ステートメントの場合は，そのステートメントが定義しているブロックと，そのブロックに含まれているブロックに対してだけ適用される。再定義をしている接頭語の有効範囲の外に制御が渡ると，その再定義は適用されなくなる。条件接頭語は，DECLARE，DEFAULT，またはENTRYステートメント以外であれば，どのようなステートメントにでもつけることができる。

ONステートメント

どのような条件に対してもシステム処置が用意されており，その条件に対するONステートメントの中でかわりの処置をプログラマーが指定していない限り，システム処置が実行される。ONステートメントの目的は，解禁されている例外条件（省略時解釈または条件接頭語のどちらかによって）が発生したために割込みが起ったときにとるべき処置を設定することである。

注：条件が禁止されているプログラム部分の実行中は，ONステートメントで指定されている処置はとられない。

ONステートメントの形式は，次の通りである。

```
ON 条件名〔SNAP〕
{オン·ユニット|SYSTEM;}
```

（詳細については，J節「ステートメント」をみていただきたい。）

セミコロンを後につけたSYSTEM というキーワードは，割込みが起ったときにはつねに標準のシステム処置をとるよう指定するものである。これは，何か他の処置がすでに設定されている条件に対して，システム処置を設定し直すものである。これは，オン・ユニットを使用して，割込みが起ったときにつねにとるべき代替の処置を指定するのに用いられる。

SNAPオプションは，割込みが起ったときに，現在のブロックを導き出す呼出しのチェインの中にあるすべてのブロックのリストを標準システム・ファイルSYSPRINT に書き出すことを指定するものである。SNAPが指定されているときは，SNAPオプションの処置が，オン・ユニットの処置よりも前にとられる。SNAP SYSTEMと指定されているときは，システム処置メッセージのすぐ後に，活動中のブロックのリストがつづけられる。

オン・ユニットは，ラベルのない，単一の単純ステートメントか，ラベルのない開始ブロックのどちらかでなければならない。この場合の単一のステートメントは，RETURN，FORMAT，DECLARE，またはDEFAULTステートメントであってはならない。また，これは，2つの複合ステートメントIFとON，あるいはDOグループであることも許されない。（PROCEDURE，BEGIN，END，およびDOステートメントは，単一のステートメントとして出てくることは許されていない。）開始ブロック（があるときは）には，どのようなステートメントでも含めることができる。ただし，RETURN ステートメントは，どのBEGIN ブロックの場合もそうであるように，開始ブロックの中にネストされているプロシージャーの中にしか置くことができない。

単一ステートメントのオン・ユニット，または開始ブロックのオン・ユニットは，プログラム内の，割込みが起った地点で呼び出されたプロシージャー（パラメーターをもたない）と同じようにして実行される。オン・ユニットが単一ステートメントのときは，それは単一ステートメントのプロシージャーであるかのように取り扱われる。そのユニットの実行が終ると，通常，そこからオン・ユニットに入ったブロックに制御が戻される。プロシージャーの場合と全く同様に，GO TOステートメントを用いて，オン・ユニットの外に制御を渡すこともできる。この場合，制御は，GO TOで指定された地点に移り，正常な戻りは行なわれない。

注：オン・ユニットから制御が戻される特定の地点は，様々な条件によってそれぞれ違っている。ある場合は，その条件が起った処置の直後の地点に戻る。また，実際の割込みの地点に制御が戻って，その処置をやり直す場合もある。ON CONVERSIONステートメントから戻る場合などは，後者の例である。変換エラーの結果，割込みが起ったときは，エラーを起した文字の変換をもう一度行なうために，オン・ユニットから制御が戻る（オン・ユニットの実行中に，誤った文字が変えられたものとして）。誤りの文字が変えられないときは，ERROR 条件が起る。

ナル・オン・ユニット

オン・ユニットの特殊な例として，ナル・ステートメントがある。これは，CONVERSIONおよびAREA 条件の場合は，再試行をしない点をのぞけば，開始ブロックのオン・ユニットからの正常な戻りと同じ効果をもっている。

ナル・オン・ユニットを使うということは，次のような2つの理由から，禁止することとは違っている。第1に，ナル・オン・ユニットは，どのような条件に対しても指定することができるが，条件は，すべて禁止できるものばかりとは限らない。第2に，条件を禁止したときは（禁示できるものの場合），この条件のチェックをなくすことによって，時間を節約することができる。ナル・オン・ユニットが指定されているときは，システムは，その条件が発生していないかどうかやはりチェックしなければならない。その結果とられる処置は，オン・ユニットから正常に戻るときにとられる処置と同じである。

注： このコンパイラーでは，誤った文字がオン・ユニットの中で変えられない限り，変換のやり直しは行なわれないので変換エラーが起ったときに，CONVERSION条件に対するナル・オン・ユニットによって，永久的なループがひき起されることはない。誤った文字が変更されなかった場合は，ERROR 条件が起こる。

ON ステートメントの有効範囲

ONステートメントが実行されると，指名された条件と処置の指定とが結びつけられる。この結びつけが一旦行なわれたあとは，それが置き換えられるか，またはそのONステートメントが実行されたブロックが終了するまで，そのままの状態が保たれる。

設定された割込みの処置は，ブロックからそれを活用化したブロックに渡り，さらにその処置は，同じ条件に対する別のONステートメントを実行することによって置き換えられない限り，後続の活用化されたブロックのすべてに対してそのまま効力をもち続ける。置き換えられる場合，新しい処置は，そのブロックが終了するまで，またはその条件に対してREVERT ステートメントが実行されるまでの間だけ，効力をもつ。それで活用化したブロックに制御が戻ると，その地点で存在していた，割込み処置の設定はすべて，設定し直される。したがって，サブルーチンは，それを呼び出したブロックに対して設定された割込み処置を変えることができないわけである。

同じ条件に対して2つ以上のONステートメントが，同一ブロック内にある場合は，後で実行された各々のONステートメントが，前に設定された条件を永久的に置き換える。同じ処置を指定している別のONステートメントを実行する（かまたは，何らかの方法で制御を移すことによって，置き換えられたONステートメントを再び実行する）以外に，再設定をすることはできない。

動的に受けつがれるオン・ユニット

あるオン・ユニットの実行中に条件を引き起して，さらにその先のオン・ユニットに入ることもできる。別のオン・ユニットの中で起ったか，または信号が送られた条件によって入ったオン・ユニットは，動的に受けつがれるオン・ユニットである。動的に受けつがれるオン・ユニットから正常に戻ると，その条件が起ったときのオン・ユニットの実行が設定し直される。

ファイル・パラメーターおよびファイル変数に対するオン・ユニット

引数およびパラメーターとして使用されるファイル定数またはファイル変数は，入力または出力条件のオン・ユニットの中で指定することができる。次の例は，この機能の規則と用法を示したものである。

例 1：

1. 別のブロック内での特定の条件に対するいくつかのオン・ユニットは，同じファイルに対して別のファイル識別名を指定することができる。たとえば，

```
E:PROCEDURE;
   DECLARE F1 FILE;
   ON ENDFILE (F1) GOTO L1;
   CALL E1 (F1);
         •
         •
         •
E1:PROCEDURE (F2);
    DECLARE F2 FILE;
    ON ENDFILE (F2) GO TO L2;
    READ FILE (F1);
    READ FILE (F2);
    END E;
```

E1の中でF2に対するファイルの終りに出あうと，E1の中にあるF2に対するオン・ユニットに入る。E1の中でオン・ユニットが指定されていない場合は，F1またはF2に対するファイルの終り条件に出あうと，Eの中のF1に対するオン・ユニットに入る。

例 2：

同じブロックの中にある特定の入力または出力に対するいくつかのオン・ユニットは，同じファイルに対して別のファイル識別名を指定することができる。2番目のオン・ユニットがあると，最初のものはこれによって置き換えられる。たとえば，

```
E: PROCEDURE;
   DECLARE F1 FILE;
   CALL E1 (F1);
        .
        .
        .
E1: PROCEDURE (F2);
   DECLARE F2 FILE;
   ON ENDFILE (F1) GOTO L1;
   READ FILE (F1);
        .
        .
        .
   ON ENDFILE (F2) GOTO L2;
   READ FILE (F2);
        .
        .
        .
   READ FILE (F1);
        .
        .
        .
   END E;
```

2番目のREAD FILE (F1); ステートメントを実行した結果，ファイルの終り条件が起ると，F2に対するオン・ユニットに入る。

例 3:

ファイル・パラメーターを指定している，特定の条件に対するREVERTステートメントが実行されると，そのファイル・パラメーターに対応する引数に対して前に設定されたオン・ユニットに入る。たとえば，

```
E: PROCEDURE;
   DCL F1 FILE;
   ON ENDFILE (F1);
   CALL E1 (F1);
        .
        .
        .
E1:  PROCEDURE (F2);
     DECLARE F2 FILE;
     ON ENDFILE (F2) GOTO L;
        .
        .
        .
     REVERT ENDFILE (F2);
  /*NULL ON-UNIT IN E ASSOCIATED
  WITH ENDFILE INTERRUPTS FOR F2*/
     READ FILE (F2);
        .
        .
        .
     END E;
```

F2に対するREADステートメントの実行中にファイルの終り条件に出あっても，E1の中のF2に対するオン・ユニットには入らない。REVERTステートメントがあるために，外側のプロシージャーの中にあるF1に対するオン・ユニットに入る。

ファイル変数を使用しているときはつねに，その効果は，変数の現在のファイル定数値を用いたときと同じになる。したがって，ファイル変数を指定するONステートメントをもつオン・ユニットは，オン・ユニットが設定されたときの変数の値であるファイル定数を指すことになる。たとえば，

```
DECLARE FV FILE VARIABLE,
        FC1 FILE,
        FC2 FILE;
FV = FC1;
ON ENDFILE (FV) GO TO FIN;
        .
        .
FV = FC2;
READ FILE (FC1);
READ FILE (FV);
```

最初のREADステートメントの間に起ったファイルの終り条件は，オン・ユニットがファイルFC1を指しているので，オン・ユニットに入る。しかし，2番目のREADステートメントでこの条件が起ったときは，このREADはファイルFC2を指しているので，オン・ユニットには入らない。

ファイル変数を指定するONステートメントが2回以上実行され，しかもそのたびに変数の値が違っているときは，実行のたびに別のオン・ユニットが設定される。次の例について考えてみよう。

```
DECLARE FV FILE VARIABLE,
        FC1 FILE,
        FC2 FILE;
        .
        .
DO FV = FC1,FC2;
   ON ENDFILE (FV) GO TO FIN;
END;
```

REVERTステートメント

REVERTステートメントは，REVERTステートメントが実行されたのと同じブロックの中で実行された指名の条件に対する最新のONステートメントの処置指定を打ち消すために，用いられる。その結果，そのブロックを活用化した時点で効力をもっていた処置が再設定される。このステートメントは，次のような形式をしている。

REVERT 条件名;

指名の条件に対してオン・ユニットが設定されていないブロックの中で実行されたREVERTステートメントは，ナル・ステートメントとして扱われる。

SIGNALステートメント

プログラマーは，SIGNAL ステートメントを用いて，ON条件の発生をシミュレートすることができる。指名の条件が禁止されていない限り，割込みが起こる。このステートメントは，次のような形式をしている。

```
SIGNAL 条件名;
```

SIGNAL ステートメントが与えられると，指定された条件に対して現在設定されている割込み処置が実行される。このステートメントは，プログラム・チェック中に，オン・ユニットの処置をテストするため，および正しい処置が条件と結びつけられているかどうか判断するために，主として用いられる。

SIGNAL で指定された条件が解禁されていない場合は，SIGNAL ステートメントは，ナル・ステートメントとして扱われる。

CONDITION 条件

次の形式のON条件を用いて，プログラマーは，

```
CONDITION(識別名)
```

CONDITION およびその識別名を指定するSIGNALステートメントが実行されたときに，つねに実行すべきオン・ユニットを設定することができる。

デバッギングの補助手段としてこの条件を使用すれば，それが実行されたときには，プログラムの現在の状況を示す情報を印刷するようなオン・ユニットを設定することができる。この手法を用いる利点は，オン・ユニットのステートメントを1度だけ書けばよいということである。これらは，プログラムの任意の地点にSIGNALステートメントを置くことによって，実行することができる。次の例は，プログラムの中にどのようにしてCONDITION条件を入れるかを示したものである。

```
ON CONDITION (TEST) BEGIN;
                        •
                        •
                        •
                       END;
```

次のステートメントが出てくると，それがどこにあっても，開始ブロックが実行される。

```
SIGNAL CONDITION (TEST);
```

この識別名は，文脈的に宣言されたものであっても（上の例のように），明示的に宣言されたものであってもよい。条件名の明示宣言は，次のようにして行なう。

```
DCL CNAME CONDITION;
ON CONDITION(CNAME) BEGIN;
                      •
                      •
                      •
                     END;
```

CONDITION 条件はつねに解禁されているが，SIGNAL ステートメントによってのみ引き起すことができる。

CHECK条件

CHECK 条件は，プログラム・テスト用にPL／Iで用意されている重要な手段である。プログラムの実行中に，指定の変数の値が修正されたとき，あるいは指定のラベルまたは入口定数が前につけられたステートメントに制御が移ったときはつねに，この条件が引き起される。CHECK 条件を解禁するためのCHECK 接頭語といっしょに与えられた，任意選択の名前リストの中で，CHECK条件を引き起すべき変数，ラベル定数，および入口定数を明示的に指定する。名前リストをつけずにCHECK 接頭語が与えられているときは，そのCHECK接頭語の有効範囲内にあるすべての変数，ラベル定数，および入口定数は，CHECK 条件を引き起すことができる。

CHECK 条件を引き起すことのできる変数には，配列変数，構造変数，ラベル変数，入口変数，事象変数，エリア変数，ファイル変数，基底付変数，定義された変数，およびロケーター変数が含まれる。添字付の変数およびロケーター修飾された変数は許されないが，修飾された名前（つまり，構造のメンバー）は使用することができる。iSUB 定義の変数は，使用できない。

チェックされる変数に対する割当ての直後に，割込みが起こる。たとえば，チェックされる配列の各要素が割り当てられると，割込みが起こる。ただし，次の場合は例外である。

1. CALL ステートメントの中で指定されている引数が直接渡される場合（見かけ引数を使って渡される場合の逆）は，サブルーチンから戻ったときに，これらの名前に対するCHECKが起こる。

2. ラベルおよび入口定数の場合は，そのステートメントの実行の直前，または入口名の呼出しの直前に割込みが起こる。

CHECK 条件に対するシステム処置は，その割込みを起した識別名を印刷すること，およびそれが変数（プログラム制御変数はのぞく）であるときは，データ指示出力の形式で，その新しい値を印

刷することである。ラベル変数および他のプログラム制御変数の場合は，識別名だけが，セミコロンで区切られて印刷される（値は含まれない）。

SIGNAL CHECKによってCHECK条件が起ったときの，標準のシステム処置は，CHECK接頭語の名前リストの中で与えられた識別名（使用できるところであれば，その値も）を印刷することである。CHECK接頭語が名前リストをもっているときの，標準のシステム処置は，CHECK接頭語の有効範囲内にあるすべての識別名（使用できるところであれば，その値も）を印刷することである。

したがって，CHECK接頭語をもつブロックを前に置くことによって（図14.1の中の例で示されているように），プログラマーは，変数を指定することにより，読みやすい形式の選択ダンプをとることができる。また，ラベルと入口名を指定することによって，流れの追跡もとることができる。

システム処置以外のものが必要な場合に，ONステートメントの中でCHECK条件を指定することもできる。これによって，ユーザーは，デバッギング情報の制御と出力用のデータ指示出力を簡単にすることも含めて，PL/Iのすべての機能を使うことができる。

SUBSCRIPTRANGE条件

主としてプログラムのチェックアウトに用いられるが，生産的なランにも使用できるもう1つのON条件は，SUBSCRIPTRANGE条件である。この条件は，条件接頭語の中で解禁しなければならない。解禁されているときは，配列参照の中の各々の添字は，評価されるたびに，それが指定の限界内にあるかどうかチェックされる。添字が高すぎるか，低すぎるときは，条件が引き起される。

このチェックには，記憶域と実行時間の両面においてかなり多くのオーバーヘッドが含まれるので，通常は，プログラムのテストにだけ用いられる－－生産的なプログラムでは，SUBSCRIPTRANGEを普通の禁止条件にして，取り外される。

STRINGRANGE条件

STRINGRANGE条件は，条件接頭語の中に書かれているのでない限り，解禁されない。この条件は，SUBSTR組込み関数および擬似変数に対して誤った参照が行なわれたときに引き起される－－この関数および擬似変数に対する引数は，特定の限界内になければならない（G節「組込み関数と擬似変数」の中の「SUBSTRストリング組込み関数」の項を参照のこと）。この条件を指定すると，自動的か（つまり，標準のシステム処置によって），またはプログラマーがオン・ユニットを使用することによって，SUBSTRの参照をつづけてゆくことができる。

STRINGRANGE 条件には，かなり多くのオーバーヘッドが含まれるので，デバッギング中にだけ解禁する方がよい。

STRINGSIZE 条件

STRINGSIZE条件は，条件接頭語の中に書かれているのでない限り，解禁されない。原ストリングを目標ストリングに割り当てた結果，目標ストリングの中で調節することのできない原ストリングの部分が切り捨てられたときに，STRINGSIZEが起こる。原ストリングと目標ストリングは，固定長または長さの変わるストリングのどちらであってもよい。この条件は，オン・ユニットといっしょに使用しても，使用しなくてもよい。標準のシステム処置は，注釈を印刷し，切り捨てられたストリングの割当てを完了し，処理をつづけることである。

STRINGSIZE条件には，かなり多くのオーバーヘッドが含まれるので，デバッギングのときだけ解禁する方がよい。

条件組込み関数と条件コード

オン・ユニットが呼び出されると，引数をもたないプロシージャーのようにして処理される。したがって，割込みに関する情報（条件の名前以外の）をオン・ユニットに渡すことはできない。オン・ユニットを使用する上でのプログラマーの一助として，割込みの原因を調べて，できればエラーを訂正することができるように，いくつかの特別な関数が用意されている。

これらの条件組込み関数は，オン・ユニットの中またはオン・ユニットによって呼び出されるブロックの中でだけ使用することができる。これらの関数は，G節「組込み関数と擬似変数」の中にリストされている。

条件組込み関数は，割込みの起ったプロシージャーの名前，変換割込みの原因となった文字または文字ストリング，最後に転送されたレコードの中で使用されているキーの値，などの情報を準備する働きをもつ。エラー訂正のための擬似変数として使用できるものもある。

ONCODE組込み関数は，最後の割込みの原因によって決まる値をもつ2進整数を与えるものである。この関数をERROR条件とあわせて用いることによって，プログラマーは，PL/I言語の

```
(CHECK(HEADER,NEWBATCH,INPUT,BADBATCH,SAMPLE)):    /*DEBUG*/
    DIST: PROCEDURE;
          DECLARE 1 SAMPLE EXTERNAL,
                      2 BATCH CHARACTER(6),
                      2 SNO PICTURE '9999',
                      2 READINGS CHARACTER(70),
                    TABLE(15,15) EXTERNAL;
   /* ESTABLISH INTERRUPT ACTIONS FOR ENDFILE & CONVERSION */
          ON ENDFILE (PDATA) CALL SUMMARY;
          ON CONVERSION BEGIN; CALL SKIPBCH;
                               GO TO NEWBATCH;
                        END;
          ON ERROR DISPLAY(BATCH||SNO||READINGS);
   /* MAIN LOOP TO PROCESS HEADER & TABLE */
  HEADER: READ INTO (SAMPLE) FILE (PDATA);
          /* CHECK ACTION LISTS INPUT DATA FOR DEBUG */
          IF SNO ¬= 0 THEN SIGNAL CONVERSION;
NEWBATCH: GET LIST (OMIN,OINT,AMIN,AINT) STRING (READINGS);
          TABLE = 0;
          CALL INPUT;
          CALL PROCESS;
          GO TO HEADER;
   /* ERROR RETURN FROM INPUT */
BADBATCH: SIGNAL CONVERSION;
(CHECK(IN1,IN2,ERR2,ERR1,TABLE)):          /*DEBUG*/
           INPUT: PROCEDURE;
           /* ESTABLISH INTERRUPT ACTIONS FOR CONVERSION & SUBRG */
                  ON CONVERSION BEGIN;
                                 IF ONCODE = 624 & ONCHAR = ' '
                                    THEN DO; ONCHAR = '0';
                                             GO TO ERR1;
                                         END;
                                    ELSE GO TO BADBATCH;
                                 END;
                  ON SUBSCRIPTRANGE GO TO ERR2;
           /* LOOP TO READ SAMPLE DATA AND ENTER IN TABLE */
             IN1: READ INTO (SAMPLE) FILE (PDATA);
                  IF SNO = 9999 THEN RETURN;      /*TRAILER CARD*/
             IN2: GET EDIT (R,OMEGA,ALPHA)(3 P'999')
                      STRING (READINGS);
(SUBSCRIPTRANGE): TABLE((OMEGA-OMIN)/OINT,(ALPHA-AMIN)/AINT) = R;
                  GO TO IN1;
           /* FIRST CONVERSION & SUBSCRIPTRANGE ERROR IN THIS BATCH */
            ERR2: ON SUBSCRIPTRANGE GO TO BADBATCH; /*FOR NEXT ERROR*/
                  CALL ERRMESS(SAMPLE,02);
                  GO TO IN1;
            ERR1: REVERT CONVERSION;    /*SWITCH FOR NEXT ERROR*/
                  CALL ERRMESS(SAMPLE,01);
                  GO TO IN2;
                  END INPUT;
           END DIST;
```

図１４.１　プログラム・チェックアウト・ルーチン

条件名では表わすことができないが，システムでは検知されるようなエラーの処理をすることが可能になる。また，同じ条件名をもつ種々の条件の区別をする－－たとえば，各種のＫＥＹ条件の区別，など－－ために使用することもできる。

ON 条件の使用例

図１４.１に示されている例は，ＯＮステートメント，SIGNALおよびREVERT ステートメント，および条件接頭語の用法を示したものである。このルーチンは，テスト用の読込みを含むカードのバッチを読むためのものである。各バッチは，ＳＮＯという名前のサンプル番号がゼロの見出しカードと，同じくＳＮＯが９９９９の後書きカードをもっている。変換エラーがみつかると，エラーの文字

セットをゼロにして，再試行が1回行なわれる。添字を計算するために2つのデータ・フィールドが使用されているが，添字が範囲外のものであれば，サンプルは無視される。2つ以上の添字エラーがあった場合，または1つのバッチの中で変換エラーが2回以上あった場合は，そのバッチは無視される。

各行の右側の番号は，カード順序番号であって，プログラム自体の一部分ではない。

カード1および25のCHECK 接頭語は，デバッギングの一助とするために組み入れられたものであり，生産的なプログラム・ランでは，取り除かれる。カード1の接頭語は，ステートメントHEADER，NEWBATCH，およびBADBATCH が実行される直前，プロシージャーINPUT が呼び出される直前，および変数SAMPLE の要素の値が変わったときはつねに，割込みを起こすよう指定している。CHECK 条件に対してONステートメントが実行されていないので，システム処置がとられる。この結果，該当する名前がSYSPRINTに書き出されることになる(SAMPLE の場合には，新しい名前といっしょに)。

内部プロシージャーINPUT の中で使用されているラベルは，DIST の中では知られていないので，DIST に対するCHECKリストの中で指定することはできない。したがって，INPUTいう見出しのPROCEDUREステートメントの直前に，別のCHECK 接頭語が挿入されている。このチェック・リストは，INPUT の中のいくつかのラベルと配列TABLE を指定している。

最初に実行されるステートメントは，カード9の中のON ENDFILE ステートメントである。これは，ENDFILE割込みが起ったときに，外部プロシージャーSUMMARY を呼び出すことを指定するものである。この処置は，DISTの中，INPUTの中およびDISTによって呼び出されたすべての他のプロシージャーの中で適用される(ただし，これらがENDFILE に対して自分自身の処置を設定している場合をのぞく)。

プロシージャー全体を通じて，SIZE，SUBSCRIPTRANGE，STRINGRANGE，STRINGSIZE，およびCHECK以外の条件は，省略時解釈によって解禁される。また，ONステートメントの中で明示的に記述されている条件以外のすべての条件に対して，システム処置が適用される。多くの場合，このシステム処置は，メッセージを生成して，ERROR 条件を引き起すことである。カード13の中でERROR 条件に対して指定されている処置は，処理されるカードの内容を表示することである。ERROR 処置が完了すると，FINISH 条件が起こり，そのあとでプログラムの実行が終了する。

カード10の中のステートメントは，COVERSION割込みが起ったときにつねにとるべき処置を指定している。この処置は，2つ以上のステートメントから構成されているので，BEGIN とEND の両ステートメントによって囲まれている。

プログラムの主ループは，HEADER というステートメントから始まる。HEADER に対してCHECK 条件が解禁されているので，HEADER が実行される前に割込みが起こる。INTO オプションをもつREAD ステートメントが実行されると，変数SAMPLE の各要素に対してCHECK 条件が引き起される。したがって，データ指示出力の形式で入力がリストされる。

読み取られたカードは，見出しカードとみなされる。そうでない場合は，SIGNAL CONVERSION ステートメントによってBEGINブロックが実行され，そこで，見出しカードに達するまでカードを無視して読み進めるプロシージャー(ここには示されていない)が，順に呼び出される。この開始ブロックは，オン・ユニットを終了させるGO TO ステートメントによって終る。

NEWBATCH というラベルのGETステートメントは，文字ストリングREADINGS(SAMPLEの一要素)に読み込まれた別のテスト番号を得るために，STRING オプションを使用している。データ・リストの中で指名されている変数は，明示的に宣言されていないので，これらが出てくると，FLOAT DECIMAL(6)という属性をつけて，暗示的な宣言が行なわれる。

プロシージャーINPUT が呼び出される前に，配列TABLEはゼロに初期値化される。このプロシージャーは，DISTの中ですでに設定されたオン・ユニットを受けつぐが，それらを打ち消すこともできる。

INPUT の最初のステートメントは，CONVERSION 割込みに対する新しい処置を設定するものである。割込みが起ったときはつねに，ONCODE がテストされて，その割込みが，誤ったP形式入力文字によるものであるか，また，誤った文字はブランクであるか，チェックされる。誤りの文字がブランクのときは，ゼロで置き換えられ，制御がERR1 に移される。

ERR1は，プロシージャーINPUT の内部にある。REVERT CONVERSION というステートメントは，INPUTの中で実行されるON CONVERSION ステートメントを無効にし，DIST の中で変換割込みに対して指定されている処置(そのバッチを無視する) を復元するものである。

エラー・メッセージを書き出すためにルーチンが呼び出されたあと，制御はIN2に行き，そこで変換が再試行される。別の変換エラーが起ったときの割込み処置は，カード10と11で指定されている処置である。

INPUT の中の2番目のON ステートメントは，SUBSCRIPTRANGE 割込みに対する処置を設定するものである。この条件は，割込みを起すためのSUBSCRIPTRANGE接頭語によって明示的に解禁しなければならない。割込みが起こると，オン・ユニットは制御をERR2に移されるが，ここでは，SUBSCRIPTRANGE 割込みに対する新しいオン・ユニットが設定されて，カード35のONステートメントで指定された処置

が打ち消される。したがって，このバッチの中でのちに添字エラーが起こると，制御はBADBATCHに移されるが，ここでは，プロシージャーDISTの中にCONVERSION条件が存在しているとき，この条件がシグナルされる。INPUTをはなれるとき，オン処置は，DISTの中で設定されている処置（この場合は，次の見出しカードを得るためにSKIPBCHを呼び出す）に戻ることに注意していただきたい。

新しいオン・ユニットが設定されたあと，メッセージが印刷され新しいサンプル・カードが読まれる。

IN1というラベルのステートメントは，80桁のカード・イメージを，構造SAMPLEに読み込む。READステートメントは，入力データの妥当性検査をしないので，CONVERSION条件が起こることはない。

ステートメントIN2は，ピクチャー形式項目にしたがって，カード桁11から19までのデータをチェックし，編集する。これらの桁に非数字（ブランクも含めて）があると，変換割込みが起こり，上で述べたような結果になる。

次のステートメント（カード41）は，SUBSCRIPTRANGE接頭語をもっている。いま読みとられたデータを用いて，二重の添字が計算される。どちらかの添字が，TABLEに対して宣言された限界の外にある場合は，割込みが起こる。両方とも限界外にあるときは，2つの割込みが起こる。

第１５章：　コンパイル時の諸機能

コンパイル時というのは，一般に，ユーザーの原始プログラムが，実行可能な目的プログラムにコンパイルされる（つまり，変換される）時間であると定義されている。通常，このときに，原始プログラムの変更をすることはできない。

しかし，ＰＬ／Ｉでは，コンパイル時にプログラマーが原始プログラムにある程度の制御を加えることができる。原始プログラムの中に，先頭のパーセント記号（％）によって識別される特別なステートメントを入れることができるが，これによって，原始プログラムの一部を，次のような幾通りかの方法で修正することができる。

1. 原始プログラムの中にある任意の識別名を変更することができる。
2. 条件付のコンパイレーションをしたい場合は，そのプログラムのどのセクションをコンパイルするかを表示することができる。
3. ユーザー・ライブラリーまたはシステム・ライブラリーの中にあるテキストのストリングを，原始プログラムに組み込むことができる。

ＰＬ／Ｉは，次のような２段階のステージを設けて，コンパイル時における原始プログラムの修正を可能にしている。

1. ブリプロセッサー・ステージ－－このステージの間に，ユーザーの原始プログラムの中にプリプロセッサー・ステートメント，（すなわち，走査されているテキストをプロプロセッサーに修正させるための特別なステートメント），があるか走査される。これらのステートメントは，原始プログラムの一部とみなされるので，このステートメントと，原始プログラムの他のテキストとを自由に入り混ぜてもよい。プリプロセッサー・ステートメントを処置した結果，修正された原始プログラムが，２番目のステージへの入力となる。プリプロセッサー・ステージは，任意選択であることに注意していただきたい。
2. プロセッサー・ステージ－－このステージの間に，最初のステージからの出力が実行可能な目的プログラムにコンパイルされる。

本章は，最初のステージに関するものであって，プログラムの実際のコンパイレーションにはふれない。

プリプロセッサー・ステートメントのほかに，プログラマーは，随意に，リスティング制御ステートメントを使うことができる。これを用いて，プログラマーは，プログラムの印刷リストのレイアウトを制御することができる。これらのステートメントは，プリプロセッサー・ステージを経由しない。

プリプロセッサーの入力と出力

プリプロセッサーは，プリプロセッサー・ステートメントを解読し，それにしたがって，原始プログラムに対して処置を加える。プリプロセッサーへの入力は，ユーザーの原始プログラムである一連の文字である。ここには，プリプロセッサー・ステートメントが，ユーザーの原始プログラムの他の部分と入り混って入っている。プリプロセッサー・ステートメントは，先頭につけたパーセント記号（％）によって識別され，プリプロセッサーがそれをみつけたときに実行される（プリプロセッサー・プロシージャーは例外で，これは，実行するためには呼び出さなければならない）。パーセント記号とステートメントの間を，１つまたは２つ以上のブランクで区切ってもよい。

プリプロセッサーは，プリプロセッサー・ステートメントを解読するほかに，注釈や文字ストリング定数の区切り符号で相手のないものがあるかどうか，原始プログラムをチェックする。また，プリプロセッサー以外のステートメントについても，正しくない文字があるかチェックする。正しくないものがあると，それをブランクで置き換える。プリプロセッサー以外のステートメントに対して，このステージで行なわれるチェックは，これだけである。

プリプロセッサーからの出力は，予備処理の済んだテキストと呼ばれる文字のストリングである。これは，修正された原始プログラムからなっており，プロセッサー・ステージへの入力となる。

プログラマーは，プリプロセッサーへの入力，およびプリプロセッサーからの出力を印刷するためのコンパイラー・オプションを指定することができる。プリプロセッサーへの入力のリスティング（プログラマーがコーディングしたプログラムを表わす）は，旧原始リスティングと呼ばれ，コンパイラーへの入力のリスティング（プリプロセッサーを使用しているときは，プリプロセッサー・テキスト）は，原始リスティングと呼ばれる。

プリプロセッサーの走査

プリプロセッサーは，ストリングの先頭から，入力テキストの走査を開始し，各文字を順々に走査してゆく。プリプロセッサー・ステートメントが出てくるまでは，各文字は，走査されたときの形式と順序の通りに，予備処理のすんだテキストに入れられる。しかし，プリプロセッサー・ステートメントが出てくると，処理が行なわれ，その結果，原始プログラムの走査と，予備処理の済んだテキストのそれ以降の作り方を，次のようにして変えることができる。

1. 実行されるステートメントが，プログラム内の別の地点から走査をつづけるよう，プリプロセッサーに命じてもよい。この新しい地点は，すでに走査のすんだ地点であってもよい。

2. 実行されるステートメントによって，置換え作業を開始することができる。つまり，プリプロセッサー・ステートメントの中に出てきていない識別名が，その後の走査中に出てきたときに，その識別名を置き換えることができる。このようにして置き換えられた値は，予備処理のすんだテキストの中の，古い識別名のあった位置に書き入れられる（詳細については，以下の「再走査と置換え」の項をみていただきたい。）

原始プログラムの最後の文字より先を走査しようとしたところで，走査は終了する。予備処理のすんだテキストが完成され，第2ステージのコンパイレーションを開始することができる。

再走査と置換え

識別名を新しい値で置き換える場合，置換えができるよう，まず，識別名を活用化しなければならない。最初に，識別名は，プリプロセッサーのDECLAREステートメント（つまり，%DECLAREステートメント）の中に出てきたときに活用化される。（%DEACTIVATEステートメントの中に出てきたときに非活用化し，%ACTIVATEステートメントの中に出てきたときに再活用化することができる）。最初に活用化されたあと，この識別名に置換え値を与えなければならない。これは通常，プリプロセッサー割当てステートメントを実行することによって行なわれる。

識別名が活用化されて，値が与えられたあとは，プリプロセッサー・ステートメント以外のテキストの中に識別名があると（走査中にこの識別名に出あったときに，それが活用状態になっていることが条件），その値によって置き換えられる。

プリプロセッサー変数は，RESCANまたはNORESCANオプションのいずれかによって活用化することができる。NORESCANオプションが適用されるときは，値は，プログラム・テキストに直接挿入される。RESCANオプションが適用されるときは，その値またはその値の一部のどちらを別の値によって置き換えるかを判定するために値がテストされている間に，再走査が行なわれる。その値を置き換えることができない場合は，その値は予備処理のすんだテキストに挿入される。その値を置き換えることができる場合は，それ以上置換えができなくなるまで，置換え作業がつづけられる。したがって，すべての置換えが終ったあとでだけ，予備処理のすんだテキストにその値が挿入される。

識別名が非活用化されると，その置換え能力は失なわれるが，その値は失なわれないことに注意していただきたい。したがって，このような識別名をのちに再活用化するときは，置換えの値を割当てる必要がない。

たとえば，原始プログラムに次のような一連のステートメントが含まれている場合は，

```
%DECLARE A CHARACTER, B FIXED;
%A = 'B+C';
%B = 2;
  X = A;
```

上記のもののかわりに，次のものが予備処理のすんだテキストに挿入される。

```
X = 2+C;
```

この例の最初のステートメントは，AおよびBを活用化するプリプロセッサーDECLAREステートメントであり，これはまた同時に，省略時解釈のRESCANによって，これらをプリプロセッサー変数として活用化もしている。2番目と3番目のステートメントは，プリプロセッサー割当てステートメントであり，2番目のステートメントは，文字ストリング'B+C'をAに割り当て，3番目のステートメントは，定数2をBに割り当てている。4番目のステートメントは，非プリプロセッサー・ステートメントであるから，このステージでは，実行されない。

（ここでは，説明しやすいように，非プリプロセッサー・ステートメントは，原始プログラムの中にあるものではあるが，プリプロセッサー・ステートメントの中にも，プリプロセッサー・プロシージャーの中にも，注釈の中にも含まれていないものであるとしよう。）

しかし，このステートメントはAを含んでおり，しかもAは，置換え用に活用化されたプリプロセッサー変数であるため，その時点におけるAの値がそのステートメントに入ることになる。したがって，ステートメントの中のAは，ストリング'B+C'によって置き換えられる。しかし，このストリングには，プリプロセッサー変数Bが含まれている。プリプロセッサーは，Bをチェックして，それがすでに活用化されたものであり，2という値が割り当てられているということを知る。したがって，ストリングの中のBは，値2によって置き換えられる。さらにチェックをつづけてゆくと，2は置き換えることができないことがわかり，+Cから走査が再開されるが，これもやはり置き換えることができない。したがって，置換えのチェインはここで終了し，その結果でき上ったステートメントが，予備処理のすんだテキストに挿入される。

上の例の中で，プリプロセッサー変数Aが，次のステートメントによって活用化されたものとすれば，

```
%ACTIVATE A NORESCAN;
```

テキストに挿入されるステートメントは，以下のようになる。

```
X = B+C;
```

これは，プリプロセッサー変数Aに対してNORESCAN が指定されているので，Aと置き換えられた結果の'B+C'の再走査が抑止されるためである。

プリプロセッサー変数Bは，省略時解釈の精度(5,0)をもっているため，実際には，2の前に4個のゼロがつけられることに注意していただきたい。この値が，ストリング'B+C'の中のBと置き換わるときは，文字ストリングに変換され，7個のブランクが前につけられた2になる(10進固定小数点数値を文字ストリングに変換するための規則にしたがう)。詳細については，この章の「プリプロセッサー式」の項を参照のこと。

置換え用の値に，パーセント記号，相手のない引用符，相手のない注釈区切り符号が含まれていてはならない。

以下の例は，コンパイル時の諸機能を用いて，DOループの実行速度を上げる方法を示したものである。

プログラマーは，次のようなループをプログラムの中に入れることができる。

```
DO I=1 TO 10;
Z(I)=X(I)+Y(I);
END;
```

次の一連のステートメントは，上の例と同じことを行なうものであるが，コンパイルされたプログラムの実行中に，増分とテストをする必要がない。

```
%DECLARE I FIXED;
%I=1;
%LAB:;
Z(I)=X(I)+Y(I);
%I=I+1;
%IF I<=10 %THEN %GO TO LAB;
%DEACTIVATE I;
```

最初のステートメントは，Iを活用化し，プリプロセッサー変数としてそれを設定するものである。2番目のステートメントは，Iに1という値を割り当てている。このことは，非プリプロセッサー・ステートメントの中でのちに識別名Iに出あったときは，1と置き換えられる(Iが活用化されていれば)ということを意味している。3番目のステートメントは，後から出てくるプリプロセッサーGO TOステートメントの転送目標として用いられるプリプロセッサーのナル・ステートメントである。

4番目のステートメントは，プリプロセッサー・ステートメントではないので，置換え作業をするために走査されるだけであり，実行はされない。このステートメントが最初に走査されるとき，Iは1という値をもっており，活用化されている。したがって，このステートメントの中にIが出てくるたびに，1と置き換えられ，形成された，予備処理ずみのテキストに次のものが挿入される。

```
Z(        1)=X(        1)+Y(        1);
```

各々の1の前には，7個のブランクがつけられることに注意していただきたい。

5番目のステートメントは，Iの値を1ずつ増やすものであり，6番目のステートメント(プリプロセッサーのIFステートメント)は，Iの値をテストするものである。Iが10より大きくなければ，LABというラベルのついたステートメントから走査が再開される。大きいときは，%GO TOステートメントの直後のテキストから走査がつづけられる。したがって，Iが10まで(10も含めて)増やされるたびに，割当てステートメントが再走査され，Iがあるたびにその時点における値と置き換えられる。この結果，次のようなステートメント群が，予備処理のすんだテキストに挿入される。

```
Z(        1)=X(        1)+Y(        1);

Z(        2)=X(        2)+Y(        2);
  .
  .
  .
Z(       10)=X(       10)+Y(       10);
```

前と同じように，1から9までの各数の前には，7個のブランクがつけられ，10の前には6個のブランクがつけられる。

Iの値が11になると，制御は%DEACTIVATEステートメントを通ることになる。このステートメントは，次のように解釈される。このあと原始プログラム・テキストの中に識別名Iがあっても，形成される予備処理ずみのテキストの中では11という値と置き換えられない。このあとの残りの走査の間中，あるいは少くとも%ACTIVATE ステートメントによってIが再活用化されるまでの間は，各々のIは修正されないままでいる。Iが再び活用化されたときは，Iはまだ11という値をもっている(間に入っていたプリプロセッサー割当てステートメントが，Iに対して新しい値を設定したのでない限り)。

プリプロセッサー変数

プリプロセッサー変数は，FIXED またはCHARACTER属性をつけて，%DECLARE ステートメントの中で指定された識別名である。プリプロセッサー変数に対して，このほかの属性を宣言することはできない。しかし，このほかの属性は，コンパイラーが与える。FIXED属性をつけて宣言されたプリプロセッサー変数には，属性DECIMAL および精度(5,0)も与えられる。CHARACTER プリプロセッサー変数には，最大の長さをもたないVARYING 属性が与えられる。プリプロセッサー・ステートメントの中で，識別名を文脈宣言，あるいは明示宣言することはできない。

プリプロセッサー変数の有効範囲は，その変数を再宣言したプリプロセッサー・プロシージャー以外のすべてのテキストにわたっている。プリプロセッサー・プロシージャーの中で宣言されたプリプ

ロセッサー変数の有効範囲は，そのプロシージャー全体である（プリプロセッサー・プロシージャーのネストがない限り）。

プリプロセッサー変数に値が与えられると，その変数が活用化されている間は，プリプロセッサー・ステートメント以外のテキストに出てきた対応の識別名はすべて，この値と置き換えられる。プリプロセッサー変数が活用化されていないときは，対応する識別名があっても置換え作業は行なわれない。

プリプロセッサー変数は，%DECLARE ステートメントの中で指定することによって，活用化したり，再活用化したりすることができる。また，%DEACTIVATEステートメントの中でこの変数を指定して非活用化し，そのあと%ACTIVATEステートメントの中で指定して再活用化することができる。プリプロセッサー変数を非活用化しても，その値は失なわれない。いいかえれば，非活用化されたプリプロセッサー変数は，それが活用化されていたときにもっていた値をそのままもっており，必要があれば，プリプロセッサー・ステートメントまたはプロシージャーによって変えることができる。

プリプロセッサー式

プリプロセッサー式は，原始プログラムの式と同じように書かれ，評価されるが，次の点だけは異なっている。

1. プリプロセッサー式のオペランドとして許されるのは，プリプロセッサー変数，プリプロセッサー・プロシージャーへの参照，10進整定数，ビット・ストリング定数，文字ストリング定数および組込み関数SUBSTR，INDEX，およびLENGTH への参照だけである。ストリング定数については，繰返し因数は許されず，組込み関数参照への引数は，プリプロセッサー式でなければならない。

2. 指数演算記号(**)は,算術演算子として使用することができない。

3. 算術演算については，精度（5,0）の10進整数算術演算だけが行なわれる。つまり，各オペランドは，演算が行なわれる前に精度（5,0）の10進固定小数点数値に変換され，10進固定小数点の結果も精度（5,0）に変換される。算術値に変換される文字ストリングは，符号付（任意）の10進整定数の形式になっていなければならない。除算記号の特性によって影響されることもあることに注意していただきたい。たとえば，3／5という式は，0.6ではなく，0と評価される。

4. 固定小数点10進数値から文字ストリングに変換すると，つねに，長さ8のストリングになる。（この数の先行ゼロは，ブランクで置き換えられ，さらに3個のブランクがこの数の左端に付け加えられるが，そのうちの1つは，その数が負のときに負符号で置き換えられる。）

プリプロセッサー変数に割り当てられる式の中にある文字ストリングには，プリプロセッサー変数，プリプロセッサー・プロシージャーへの参照，定数，および演算子が含まれていてもよい。このようなストリングに，プリプロセッサー・ステートメントを含めることはできない。

プリプロセッサー・プロシージャー

プリプロセッサー・プロシージャーは，プリプロセッサー・ステージでだけ実行可能な内部関数プロシージャーである。その統語規則が他の関数プロシージャーと違っている点は，そのPROCEDURE およびENDステートメントはそれぞれ，先行パーセント記号をもっていなければならないことである。プリプロセッサー・プロシージャーの形式は，次の通りである。

```
%ラベル：〔ラベル：〕… PROCEDURE
        〔（識別名
           〔，識別名〕…）〕
           RETURNS{（CHARACTER|FIXED）}；
                    ・
                    ・
                    ・
〔ラベル：〕…RETURN
             （プリプロセッサー式）；
                    ・
                    ・
                    ・
%〔ラベル：〕…     END〔ラベル〕；
```

RETURN ステートメントが2つ以上あってもよい。プリプロセッサー・プロシージャーの中に書くことのできるステートメントに関する一般規則については，J節「ステートメント」の中の%PROCEDUREステートメントの説明をみていただきたい。ただし，次のことだけは注意しなければならない。すなわち，プリプロセッサー・プロシージャーの中に出てくるステートメントは，先頭にパーセント記号をもっていてはならない。

プリプロセッサー・プロシージャーの呼出し

プリプロセッサー・プロシージャーは，普通の場合関数参照と呼ばれる方法で，つまり，入口名とそれに関連する引数リスト（あれば）を式の中に書くという方法で，呼び出される。関数参照は，プリプロセッサー・ステートメント，または非プリプロセッサー・ステートメントの中に書くことができる。

非プリプロセッサー・ステートメントの中で，プリプロセッサー・プロシージャーを参照するときは，その参照の中で使用される入口名は，その参照が出てきた時点で活用状態になっていなければならない。プリプロセッサー関数の入口名は，活用化と非活用化に関する限り，プリプロセッサー変数と同じである。つまり，これらは，%DEACTIVATE ステートメントまたは%ACTIVATE ステートメントによって，最初に活用化することができる。

プリプロセッサー・プロシージャーは，呼び出される前に走査される必要はない。しかし，これは，呼出しの地点で，主テキストの中または組込まれたテキスト（%INCLUDE ステートメントによって）の中のどちらかにあるものでなければならない。プリプロセッサー・プロシージャー入口名を，%DECLARE ステートメントの中で指定する必要はない。

プリプロセッサー関数から返される値（すなわち，RETURNステートメントの中のプリプロセッサー式の値）は，つねに，関数参照およびそれに関連する引数リストと置き換わる。プリプロセッサー・ステートメントの中で参照が行なわれた場合は，ステートメントのその回の実行に対してだけ，置換えが行なわれるのであって，その後そのステートメントを走査すると，再び関数が呼び出される結果になる。

プリプロセッサー関数に対する引数およびパラメーター

プロシージャー参照の中の引数の数と%PROCEDUREステートメントの中のパラメーターの数は，同じである必要はない。関数参照が出てきたステートメントのタイプ（プリプロセッサーか非プリプロセッサーか）にしたがって，引数が解釈される。引数リストの中の引数は，パラメーター・リストと突き合せられる前に評価される。パラメーターよりも引数が多い場合は，右側にある余分の引数が無視される。（関数参照の引数の場合は，後で引数が無視されても，関数は呼び出されて実行されることに注意していただきたい。）パラメーターよりも引数が少ない場合は，右側の余分のパラメーターには，FIXED パラメーターのときはゼロの値，CHARACTER パラメーターのときはナル・ストリングが与えられる。関数参照がプリプロセッサー・ステートメントの中にある場合は，見かけ引数の作成に関する通常の規則が適用されるが，関数参照が非プリプロセッサー・ステートメントの中にある場合は，つねに見かけ引数が作られる。

非プリプロセッサー・ステートメントの中に関数参照があるときは，引数は文字ストリングとして解釈され，カンマ，または対になった括弧の外側に右括弧があると，そこで区切られる。たとえば，引数リスト（A（B，C），D）は引数を2つもっている。すなわち，ストリングA（B，C）とストリングDである。そのあとで，可能な置換え作業があるか，各引数が走査される。プロシージャー名とその引数リストの両方が，1つの置換えレベルでみつからなければならない。したがって，このプロシージャー名に出あったときに走査されるテキストの中にあるカンマと括弧だけが，この文脈の中で考慮される。すべての置換えが行なわれたあと，その結果の引数は各々，その関数入口名に対するプリプロセッサー・プロシージャー・ステートメントの中の対応するパラメーター属性によって表示されているタイプに変換される。

プリプロセッサー・ステートメントの中に関数参照がある場合は，ごく普通の形式で，引数とパラメーターとが結びつけられる。一致していないときは，対応するパラメーターの属性に合わせて変換される。プリプロセッサー・ステートメントによって呼び出されるプリプロセッサー関数に渡すことができるのは，プリプロセッサー変数，文字ストリング定数，および固定小数点10進定数だけである。

返される値

プリプロセッサー関数から呼出し地点に返される値は，その関数のRETURN ステートメントの中にあるプリプロセッサー式によって表わされる。返される前に，この値は，その関数の%PROCEDUREステートメントのRETURNSオプションの中で指定された属性（CHARACTERまたはFIXED）に変換される（必要なら）。呼出し地点が非プリプロセッサー・ステートメントの中にあり，しかもその入口名がNORESCANオプションによって活用化されていないときは，関数参照を置き換えたあとで，その値は，置換え作業のために走査される。

プリプロセッサー・プロシージャーから返される値に，プリプロセッサー・ステートメントが含まれることとは，プリプロセッサー式の規則では許されないことに注意していただきたい。

プリプロセッサー関数の例

以下のステートメントの中のVALUE は，引数1（引数2）の形で文字ストリングを返すプリプロセッサー関数プロシージャーである（ただし，引数1および引数2は，この関数に渡された2つの引数を表わしている）。

原始プログラムには，次のような一連のステートメントが含まれているものとしよう。

```
%DECLARE A CHARACTER;
%DECLARE VALUE ENTRY;
DECLARE (Z(10),Q) FIXED;
%A='Z';
%VALUE: PROCEDURE(ARG1,ARG2) RETURNS
              (CHARACTER);
          DECLARE ARG1 CHARACTER,
              ARG2 FIXED;
          RETURN(ARG1||'('||ARG2||')');
        %END VALUE;
Q = 6+VALUE(A,3);
```

走査が最後のステートメントに達すると，Aが活用化されるので，Aは置換え可能になる。VALUE も活用化されているので，最後のステートメントでVALUE を参照すると，プリプロセッサーは，その名前のプリプロセッサー関数プロシージャーを呼び出す。しかし，引数Aと3がVALUE に渡される前に，Aはその値Z（前の割当てステートメントでAに割り当てられている）によって置き換えられ，3は，対応するパラメーターの属性に合わせて固定小数点に変換される。次に，VALUE は，これらの引数と括弧を連結し，連結した値，すなわちストリングZ（3）を呼出し地点に返す。返された値は関数参照と置き換わり，その結果が予備処理のすんだテキストに挿入される。したがって，上記の一連のステートメントによって生成された予備処理ずみのテキストは，次のようになる。

```
DECLARE (Z(10),Q) FIXED;
Q = 6+Z(         3);
```

次の例の中で定義されているプリプロセッサー関数プロシージャーGENは，最高99個の入口名に対して（パラメーター記述子リストの中に最高99個のパラメーター記述子を入れて）GENERIC宣言を生成することができる。しかし，この例では4個しか生成されない。

原始プログラムの中に，次のような一群のステートメントが入っているものとしよう。

```
%DCL GEN ENTRY;
DCL A GEN (A,2,5,FIXED),...;
     .
     .
     .
%GEN: PROC (NAME,LOW,HIGH,ATTR)
              RETURNS (CHAR);
   DCL (NAME,SUFFIX,ATTR,STRING)
   CHAR,(LOW,HIGH,I,J) FIXED;
    STRING='GENERIC(';
    DO I=LOW TO HIGH; /* ENTRY
                        NAME LOOP */
   IF I>9
       THEN SUFFIX=SUBSTR(I,7,2);
          /* 2 DIGIT SUFFIX*/
       ELSE SUFFIX=SUBSTR(I,8,1);
          /*1 DIGIT SUFFIX*/
     STRING=STRING||NAME||SUFFIX||
       ' WHEN (';
     DO J=1 TO I; /* DESCRIPTOR
       LIST*/
       STRING=STRING||ATTR;
       IF J<I /* ATTRIBUTE
                      SEPARATOR */
          THEN STRING=STRING||',';
       ELSE STRING=STRING||')';
          /* LIST SEPARATOR */
      END;
   IF I<HIGH /* ENTRY NAME
                   SEPARATOR*/
          THEN STRING=STRING||',';
          ELSE STRING=STRING||')';
          /* END OF LIST */
    END;
   RETURN (STRING);
 % END;
```

上記のプリプロセッサー・プロシージャーでは，次のようなテキストが作られる。

```
DCL A GENERIC(A2 WHEN (FIXED,FIXED),
              A3 WHEN (FIXED,FIXED,
                FIXED),
              A4 WHEN (FIXED,FIXED,
                FIXED, FIXED),
              A5 WHEN (FIXED,FIXED,
                FIXED, FIXED,FIXED));
```

SUBSTR，LENGTH，および INDEX 組込み関数の用法

非プリプロセッサー・ステートメントの中でSUBSTR，LENGTH，または INDEXが参照されていると，これらの名前が活用化されているときだけ，プリプロセッサーはこれらの参照を実行する。これらの組込み関数は，%DECLARE または，%ACTIVATEステートメントによって活用化される。SUBSTR，LENGTH，および INDEX という識別名のいずれかが，%PROCEDURE ステートメントの接頭語としてついている場合は，その名前をもつユーザー定義のプリプロセッサー・プロシージャーが参照されているものとみなされる。このような場合は，それらの組込み関数がプリプロセッサー・プロシージャーの中で使用されるときに，識別名SUBSTR，LENGTH，および INDEXにBUILTIN 属性をつけて，宣言し直すことができる。

組込み関数は，原始テキストまたはプリプロセッサー・ステートメントの中にあるときは，ユーザーのプリプロセッサー関数と同じように働く。

SUBSTR，LENGTH，および INDEX の最初の引数は，必要に応じて文字に変換される。SUBSTR の2番目と3番目の引数は，必要なら，10進数に変換される。INDEX の2番目の引数は，必要なら文字に変換される。返される値は，SUBSTR のときはCHARACTERで，LENGTH または INDEXのときはFIXED である。

プリプロセッサーDOグループ

プリプロセッサーDOグループは，そのグループに含まれるプリプロセッサー・ステートメントの反復実行を行なうことができる。プリプロセッサーDOグループの形式は，以下の通りである。

```
%〔ラベル：〕…    DO    i=m1  TO  m2〔BY  m3〕;
                                BY  m3〔TO  m2〕
                    .
                    .
                    .
%〔ラベル：〕…    END〔ラベル〕;
```

上の形式の中で，iはプリプロセッサー変数でなければならず，m1，m2，およびm3は，プリプロセッサー式でなければならない。ENDというキー・ワードの後につけることのできるラベルは，キーワードDOの前にあるいくつかのラベルの1つでなければならない。プリプロセッサーDOグループはネストされていてもよいし，また，複数閉止も許される。

反復を指定しているプリプロセッサーDOグループに制御を移すことはできないが，そのグループから呼び出されたプリプロセッサー・プロシージャーから戻る場合は例外として許される。

プリプロセッサー・ステートメント，およびプリプロセッサー・ステートメント以外のテキストは両方とも，プリプロセッサーDOグループの中に入れることができる。しかし，実行されるのはプリプロセッサー・ステートメントだけである。非プリプロセッサー・ステートメントは，走査されるが，置換え作業が可能であるか走査されるだけである。

非反復のプリプロセッサーDOグループは，%IFステートメントのTHENまたはELSEクローズとして役に立つ。

プリプロセッサーDOグループの展開は，J節「ステートメント」の非プリプロセッサーDOステートメントのところで示されているものと同じである。

以下の例の結果は，この章の前の方の「再走査と置換え」のセクションのところでとり上げたプリプロセッサー・ループの展開の例に対して生成されたものと同じ展開になる。

```
%DECLARE  I  FIXED;
%DO  I=1  TO  10;
Z(I)=X(I)+Y(I);
%END;
%DEACTIVATE  I;
```

「返される値」のところの2番目の例は，プリプロセッサー・プロシージャーの中でプリプロセッサーDOグループをどのように使用することができるかを示したものである（もちろん，パーセント記号は省かなければならない）。

外部テキストの組込み

外部テキストのストリングは，%INCLUDE ステートメントを用いて原始プログラムに組み込むことができる。このようなテキストは，一旦組み込まれたあと，組み込まれたテキストと呼ばれ，プリプロセッサーおよび非プリプロセッサー・ステートメントの両方を入れて構成することができる。したがって，組み込まれたテキストは，予備処理されたテキストの形成に役立てることができる。

注：　必要とされるプリプロセッサー機能がテキストの組込みだけである場合，つまり，原始プログラムにも，組み込まれたテキストにも，%INCLUDE 以外のプリプロセッサー・ステートメントが含まれていない場合は，プリプロセッサー・ステージを省略してもよい。さらにくわしい点については，このコンパイラーのプログラマー手引書をみていただきたい。

組み込まれたテキストにプリプロセッサーの宣言が含まれている場合は，プリプロセッサー変数として宣言された名前の有効範囲は，組み込まれたテキスト全体と，組み込まれたテキストの後にあるテキストである（ただし，その名前が再宣言されているプリプロセッサー・プロシージャーはのぞく）。

%INCLUDE ステートメントの用法に関する一般形式と規則については，J節「ステートメント」のところに示してある。

%INCLUDE ステートメントによって指定されたテキストは，原始プログラムの中の，そのステートメントが実行される地点の直後に組み込まれる。したがって，走査は，組み込まれたテキストの最初の文字から続けられる。このテキストの中にあるプリプロセッサー・ステートメントはすべて実行され，必要なところでは置換えが行なわれる。

外部テキストの中でその宣言が行なわれているプリプロセッサー・プロシージャーは，その外部テキストが組み込まれたテキストになったあとでだけ呼び出すことができる。あるプリプロセッサー・プロシージャーが原始プログラムに組み込まれる前にそのプロシージャーが参照された場合，その結果はどうなるかわからない。

PAYROLL は，原始ステートメント・ライブラリーの中のブックで，その中に次のような構造の宣言が含まれているものとしよう。

```
DECLARE 1 PAYROLL,
        2 NAME,
          3 LAST CHARACTER (30) VARYING,
          3 FIRST CHARACTER (15) VARYING,
          3 MIDDLE CHARACTER (3) VARYING,
        2 MAN_NO FIXED DECIMAL (6,0),
          3 REGLR FIXED DECIMAL (8,2),
          3 OVERTIM FIXED DECIMAL (8,2),
        2 RATE,
          3 REGLAR FIXED DECIMAL (8,2),
          3 OVERTIME FIXED DECIMAL (8,2);
```

次の一連のプリプロセッサー・ステートメントは，

```
%DECLARE PAYROLL CHARACTER;
%PAYROLL='CUM_PAY';
%INCLUDE PAYRL;
%DEACTIVATE PAYROLL;
%INCLUDE PAYRL;
```

2つの全く同じ構造宣言（名前だけがCUM_PAYとPAYROLLというように違っている）を生成して，予備処理のすんだテキストに入れる。最初の%INCLUDE ステートメントが実行されると，PAYRL の中のテキストが原始プログラムに組み入れられる。この組み込まれたテキストの中を走査してPAYROLL という識別名に出あうと，活用化されたプリプロセッサー変数PAYROLL の現在の値，すなわちCUM_PAY によって置き換えられる。組み込まれたテキストをさらに走査しても，これ以上置換えは行なわれない。そのあとの走査では，%DEACTIVATEステートメントがみつかる。このステートメントが実行されると，プリプロセッサー変数PAYROLL が非活用化され，この識別名は置換えが不可能になる。2番目の%INCLUDEステートメントが実行されると，PAYRL の中のテキストがもう1度原始プログラムに組み入れられる。しかし，今度は，組み込まれたテキストを走査しても，組み込まれたテキストの中の識別名はいずれも活用化されていないので，置換えは何も行なわれない。このようにして，名前だけが違う2つの構造宣言が，予備処理のすんだテキストに挿入される。

プリプロセッサー・ステートメント

ここでは，プリプロセッサー・ステージで使用することのできるステートメントをあげて，本章でまだ説明していないプリプロセッサー・ステートメントについて簡単に説明する。すべてのプリプロセッサー・ステートメント，その形式，およびその用法に関する規則については，J節「ステートメント」の「プリプロセッサー・ステートメント」の項で説明する。

しかし，まず最初に，一般にプリプロセッサー・ステートメントについて注意すべきことをいくつかあげておかなければならない。

1. プリプロセッサー・ステートメントの中に出てくるキーワードのうちのあるものは，C節「キーワードとその省略形」のところで示されているように，略語にすることができる。

2. ブランクを入れることのできる場所であれば，プリプロセッサー・ステートメントの中に注釈を入れることができる。しかし，このような注釈は，予備処理のすんだテキストには挿入されない。

3. プリプロセッサー・ステートメントはすべて，ラベルをつけることができる。このようなラベルは，パーセント記号の直後（ブランクだけは，間にあってよい）に置かなければならない。ラベルはすべて，添字のつかないステートメント・ラベル定数でなければならない。（%DECLARE ステートメントについているラベルは無視される。）

次にあげるプリプロセッサー・ステートメントによって行なわれる機能については，この章ですでに説明した。

```
%ACTIVATE
%DEACTIVATE
%DECLARE
%DO
%END
%INCLUDE
%PROCEDURE
RETURN
```

プリプロセッサーのRETURN ステートメントは，プリプロセッサー・プロシージャー内部でしか使うことができないものであり，その中ではパーセント記号を省略しなければならないので，先頭にパーセント記号をつけることはできないことに注意していただきたい。

以上のほかに，プリプロセッサー・ステージで実行できるステートメントが4つある。

```
%割当て
%GO TO
%IF
%ナル
```

プリプロセッサー割当てステートメントは，プリプロセッサー式を評価し，プリプロセッサー変数にその結果を割り当てるのに，用いられる。このセクションで示されている例はすべて，このステートメントを使用している。

%GO TOステートメントは，順次的な走査を中断して，原始プログラム内の別の場所（特に，%GO TOの中で指定されたラベルの位置）から走査をつづけるよう，プリプロセッサーに指示するものである。したがって，これは，テキストを再走査したり，飛ばしたりするときに有用である。

%IFステートメントは，プリプロセッサー式の値に応じて，走査の順序を制御するのに用いることができる。このステートメントの中でTHENクローズは必須のものであり，ELSEクローズはあってもなくてもよい。どちらのクローズにも，また，そのクローズの中のプリプロセッサー・ステートメントにもそれぞれ，先頭にパーセント記号をつけなければならない。%IFステートメントをネストすることができるが，その場合には，非プリプロセッサーIFステートメントのネスティングに対して適用されたものと同じ規則にしたがわなければならない。

プリプロセッサーのナル・ステートメントは，パーセント記号がつく点以外は，非プリプロセッサーのナル・ステートメントと同じである。これは，%GO TOステートメントの行先を設定したり，ネストされた%IFステートメントの中で%ELSEクローズのバランスをとったりするために，使用することができる。たとえば，%ELSE%；は，ナルのELSEクローズである。

プログラマーのコーディング	非原始リスティングまたは原始リスティング（プリプロセッサーをもたない場合）	予備処理後の原始リスティング
A=B； %SKIP(2)； C=D；	A=B； C=D；	A=B； C=d；
X=2； %SKIP(1)； Y=0；	X=2； %SKIP(1)； Y=0；	X=2； Y=0；
B1='1'B； %SKIP；B2='0'B；	B1='1'B； %SKIP；B2='0'B；	B1='1'B； B2='0'B；
P=0；…%SKIP(1)；SIG NAL CONVERSION； （行の最後の桁に,SIGNAL のGがある）	P=0；…%SKIP(1)；SIG NAL CONVERSION；	P=0；… SIGNAL CONVERSION；
RES=SQRT(X)； %PAGE； DCL Z FLOAT；	DCL Z FLOAT； 新しいページの先頭	DCL Z FLOAT； 新しいページの先頭
OPEN FILE (F1)； %PAGE； PUT (REC_1)；	新しいページにスキップしない。	PUT (REC_1)； 新しいページの先頭
END LOOP_3B;%PAGE;A LLOCATE A_3； （行の最後の桁にALLOCATEのAがある）	新しいページにスキップしない。	ALLOCATE A_3； 新しいページの先頭

図15.1 %PAGE および%SKIP ステートメントの効果

リスティング制御ステートメント

印刷されるリスティングのレイアウトをプログラマーが制御するために，次のようなステートメントが2つある。

%SKIP〔(n)〕；
%PAGE；

様々な状況における%SKIPおよび%PAGE ステートメントの効果が，図15.1に示されている。

%SKIP は，リスティングの中で何行かをスキップすることを指定するものであり，%PAGE は，次のページからリスティングをつづけることを指定するものである。

どちらのステートメントも先頭にパーセント記号をもってはいるが，必ずしもプリプロセッサーの使用を必要とするわけではない。しかし，プリプロセッサーを使用しているときは，これらのステートメントは，旧原始リスティング（プリプロセッサーへの入力）にも，原始リスティング（予備処理のすんだテキスト）にも適用される。

旧原始リスティングか原始リスティングのいずれかに，%SKIP(n)ステートメントを適用すると，リスティングの中で次の行を印刷する前に，n行のブランク行が挿入される。%SKIP；というステートメントは，%SKIP(1)；と同じである。

%PAGE；というステートメントは，リスティングの中のテキストの次の行を，次のページの最初の行に印刷させるものである。%PAGE ステートメントの中で指定できるオプションはない。

%SKIP または%PAGE ステートメントが，効力をもったあとは，プリプロセッサーはこれらを印刷しないし，またコンパイラーがこれらをテキストから削除する。したがって，形式化されたリストには出てこない。

リスティングの形式化をするために，リスティング制御ステートメントは，他のステートメントとは別にしてテキストの1行に書かなければならない。

プリプロセッサーが使用されているときは，同じ行にテキストをもつ%SKIP または%PAGE ステートメントは，プリプロセッサーによって，それ自身の行に移され，コンパイラー・リスティングが印刷されるときにそれが効力を発するようにされる。したがって，プリプロセッサー・リスティングは，コーディングされた通りに印刷されるが，原始リスティングは，適当に行またはページがスキップされた形になる。

プリプロセッサーが用いられているときは，%PAGE または %SKIP が含まれている行の終りで分割されているキーワードや他の識別名は，2つの部分が連結されて，原始リスティングの中では完全な識別名を形成するようになる。%SKIP または%PAGE を削除することによって充分なスペースが得られた場合は，最初の部分が出ている行に完全な語が印刷されるが，得られない場合は，新しい行が開始される。

SKIP(n); ステートメントのnが，そのページに残っている行数よりも大きいときは，%PAGE ステートメントと同じものがかわりに実行される。

第16章： 最適化と実行効率

この章は，3つのセクションにわかれている。最初のセクションでは，最適化機能について説明する。2番目のセクションには，プログラムの効率を上げるための実際のコーディングをいくつか上げてある。3番目のセクションでは，初めてPL／Iを使用するプログラマーが陥いりやすい誤りや落し穴をいくつか紹介する。

最適化

最適化の主な目的は，できるだけ速く実行され，しかも実行中に占めるスペースができる限り少ない目的プログラムを生成することである。多くの場合，これは，プログラマーの書いたステートメントに対して効率のよいコードを生成するということを意味しているが，その他にも，同じ結果を作り出す一方で効率を上げるために，ステートメントや操作の順序を，コンパイラーが変えることもできる。

最適化コンパイラーでは，次のようなタイプの最適化が行なわれる。

- 共通の式を排除する。
- 不変の式をループから移動する。
- 余分な式を排除する。
- 式を単純化する。
- 配列と構造を初期値化する。
- 変換をインライン・コードにする。
- レコード入出力転送ステートメントをインライン・コードにする。
- REGIONALデータ・セットの場合のキー変換を減らす。
- 形式リストとデータ・リストを突き合わせる。
- ストリングの取扱いをインライン・コードにする。
- 大半の組込み関数をインライン・コードにする。
- DOステートメントに対して特別な場合のコードを作る。
- 配列と構造に対して特別な場合のコードで作る。
- できるだけ長い間値をレジスターにとゞめておき，最適の流れ経路にもとづいて効率のよいアドレス計算を行うことによって，レジスターとアドレスを最適化する。
- できる限り，プログラムのブランチを同じ基底アドレスに対するものにする。
- 共通の定数とプログラム制御データを排除して，スペースの使用を最小に押える。

重要な機能のいくつかについては，次のセクションでさらにくわしく説明する。

共通式

「共通式」という語は，次の例のようにB＋Cというような式を記述する際に用いられるもので，

```
A = B+C
    :
D = B+C
```

この場合，変数BとCは，これらの2つの式の間でリセットされないものとする。このような場合は，明らかに，共通の式を1度だけ評価すればよいわけである。

同じ式を繰返して評価することを避ける手法は，共通式の排除と呼ばれる。

この手法が用いられる重要な例としては，各変数に対して同じ添字値が使用されるような，添字つきの変数が含まれるステートメントの例があげられる。たとえば，

```
PAYROLL_TAX(MANNO) = PAY_CODE(MANNO) *
  WEEKPMNT(MANNO);
```

添字式MANNO の値は，ステートメントが実行されるときに1度だけ計算される（MANNOが10進変数と宣言されているときには，10進数から2進数への値の変換もこの計算に含まれる）。

共通式の排除が行なわれているプログラムに対する割込みの取扱い

PL／I の各ステートメントの中にある殆んどの演算の順序は，そこ

に含まれる演算の優先順位にしたがって，行なわれる。しかし，優先順位の低い演算子のオペランドとなるような結果を生じる副次式（添字式，ロケーター修飾式，関数参照など）の評価の順序は，別の演算でその値を使用する前にオペランドが完全に評価されていなければならないという規則以外には，決められていない。したがってこのような副次式の評価中に起こる割込みと結びつけられるオン・ユニットには，予測できない順序で入る可能性がある。したがって，オン・ユニットに入る順序やそれによってとられる処置次第で，1つの式がいろいろな値をとる可能性がある。計算のオン・ユニットに入るときは，

1. 前のステートメントの実行によってセットされたすべての変数の値は，その変数に割り当てられた最後の値であることが保証されているので，オン・ユニットはこれを使用することができる。たとえば，オン・ユニットに入ったときにすべての変数の値を記録するために，PUT DATA ステートメントを使用することができる。

2. 計算割込みから生じたオン・ユニットの中でセットされた変数の値は，プログラムのどの部分でも，その変数に割り当てられた最後の値であることが保証されている。

関連するオン・ユニットの中で，またはオン・ユニットからのブランチを実行した結果として，起った計算割込みの結果，変数が修正される可能性がある場合には，共通式の排除は禁止される。たとえば，

```
ON ZERODIVIDE B,C=1;
   .
   .
   .
X=A*B+B/C;
Y=A*B+D;
```

コンパイラーは通常，2番目の割当てステートメントの中の副次式A＊Bの再評価を排除しようとする。しかし，この例では，B／Cの評価中にZERODIVIDE条件が起った場合は，A＊Bの2つの値が違ってくる可能性もある。この可能性を考慮して，この最適化は禁止される。

上の説明は，最適化オプションORDERが指定されているか，想定されているときにだけ適用されることに注意していただきたい。プログラマーが，上で述べた保証を必要としないときは，最適化オプションREORDER を指定することができる。この場合，共通式の排除は禁止されない。ORDERおよびREORDERオプションについては，この章の後の方で説明する。

不変の式またはステートメントの移動

ループの中に出てくる式またはステートメントは，その式の値またはステートメントの処置がループの反復のたびに同じであることをコンパイラーが発見できるときは，不変であるという。

不変の式またはステートメントを，ループの中から，ループの外側にあるプログラム内の地点へ移動して，ループを反復するたびではなく，1度だけ実行することができる。たとえば，

```
DO I = 1 TO N;
   .
   .
   .
 J = 3;
   .
   .
   .
END;
```

J＝3；というステートメントは不変であるから，ループの外へ移動することができる。状況に応じて，前に移しても後に移してもよい。

このタイプの最適化を利用したい場合は，順序変更の可能なステートメントまたは演算をもつループが含まれているBEGINまたはPROCEDUREブロックに対して，最適化オプションREORDERを指定しなければならない。このオプションが指定されていないときは，省略時解釈のオプションORDERが想定され，最適化は禁止される。

ORDERおよびREORDERオプション

ORDERおよびREORDERは，PROCEDURE またはBEGIN ステートメントの中で，プロシージャーまたは開始ブロックに対して指定される最適化オプションである。

標準の省略時解釈はORDER であるが，内に含まれるブロックはすべて，明示的にORDER が指定されていない限り,REORDERを受けつぐ。

ORDER オプション

ブロックの中で修正される変数に対して最後に割り当てられた値を，ブロックの中でステートメントおよび式を実行しているときに起った計算割込みのために入ったオン・ユニットの中で使用できるようにしたい場合は，プロシージャーまたは開始ブロックに対してORDERオプションを指定しなければならない。ORDERオプショ

ンが適用されているブロックの中では，コンパイラーが共通式を排除することがある。そのような場合は，ブロックの実行中に起こる計算割込みの回数は，共通式が排除されなかった場合にくらべて，少ないかもしれない。しかし，ORDERブロックの実行中に割込みが起った場合は，割込みの前にあるステートメントの中で変数に割り当てられた値は，割込みに対するオン・ユニットの中でそれらが参照されたとき，最後に割り当てられた値であることが保証される。ORDERブロックの中では，割込みを起す可能性のある式を前または後に移動することをのぞいて，他の形式の最適化が許されている。起こる可能性のあるすべての条件を禁止する必要があるので，ORDERを使用すると，実質的に，ループからステートメントまたは式を移動することは，抑止されることになる。

REORDERオプション

REORDERオプションを使用すると，コンパイラーはある程度の柔軟性が得られるので，生成される目的コードの中で高いレベルの最適化を行なうことができる。

ループの中に書かれているステートメントが，ループの中でそれを繰返し実行することが余分であるような場合がある。REORDERオプションを用いると，コンパイラーは，そのステートメントが一度だけ実行されるように，このようなステートメントを原始プログラムの中でループの前または後の地点に移動することができる。同様に，ループの影響に関して不変である式は，ループの外側の一地点で1度だけ評価することができる。

ループの実行中に頻繁に修正される対象となる変数の値をレジスターに保持しておくことによって，ループをより効率よく実行することができる。エラーなしで実行ができるときは，レジスターの中に値を保存することができるので，記憶位置で変数の値をリセットするための，時間のかかるロードと記憶の操作を排除することによって，大巾に効率をよくすることができる。ループが実行されたあとで変数の最新の値が必要なときは，制御がループの外に渡されたときに変数の記憶位置に値が割り当てられる。

ブロックに対してREORDERが指定されているときは，レジスターの割振りを有意義に最適化することができる。しかし，計算割込みが起ったとき，ブロックの中でリセットされる変数の値は，最後に割り当てられた値であるとは保証されない。これは，変数の最新の値はレジスターの中にはあるかもしれないが，変数の記憶位置にはないためである。したがって，計算割込みのために入るオン・ユニットでは，REORDERブロックの中でセットされる変数を参照してはならない。しかし，組込み関数ONSOURCEおよびONCHARは，この文脈の中で使用してもよい。

ステートメントの実行または式の評価の順序を変更すると，変更をしなければ，条件付のPL／Iステートメントによって避けられたはずの計算割込みが起こる可能性がある。たとえば，

```
IF Y ¬= 0 THEN DO J = 1 TO K;
               Z(J) = X/Y;
               END;
```

X／Yがループの外側で評価された場合は，Yの値がゼロのとき，ZERODIVIDE条件が起こる。

順序変更の結果，次のようなことが起こる場合は，ORDERオプションを指定しなければならない。

1. 計算割込み，または
2. 故意に初期値化しない変数を使用する。

このどちらも，順序変更がなければ，条件付のPL／Iステートメントによって避けることができる。

値が保証されていない変数がうしろで使用されているREORDERブロックの中で計算割込みまたはシステム処置割込みが実行中に起った場合は，プログラムはエラーになる。

CONVERSIONオン・ユニットに対してONSOURCEおよびONCHAR擬似変数を使用している場合をのぞいて，これらの制約は，エラーのあるデータの集りを排除するので，プログラマーは，標準のシステム処置にまかせてプログラムの実行を中止するか，あるいは，オン・ユニットを使用して，エラーの回復処置を行ない，計算のために新しいデータを得てプログラムを再開するかしなければならない。2番目の手法を用いるときは，できる限り最適化を利用しながら，すべての有効なデータを処理し，無効なデータには注をつけるようにしなければならない。

次の例は，REORDERオプションの効果を示したものである。

```
ON OVERFLOW PUT DATA;
DO J = 1 TO M;
DO I = 1 TO N;
X(I,J) = Y(I) + Z(J) *L + SQRT(W);
P = I*J;
END;
END;
.
```

上記のステートメントがREORDERブロックの中にあると，コンパイルされる原始コードは，次のように解釈される。

```
ON OVERFLOW PUT DATA;
TEMP1 = SQRT(W);
DO J = 1 TO M;
TEMP2 = Z(J) * L + TEMP1;
DO I = 1 TO N;
X(I,J) = Y(I) + TEMP2;
END;
END;
P = N * M;
```

TEMP1 とTEMP2 は，ループの後に移された式の値を入れておくために作られた一時的な変数である。P = I * J；というステートメントは，Pの値が必要になるのは外側のループの最後の反復のあとだけなので，ループの前に移動することもできる。オーバーフロー割込みが起ったときは，ループの中で使用される変数の値は，割込みが起こる前に割り当てられた最新の値であるとは保証することはできない。これは，その時点の値はレジスターには入っているかもしれないが，オン・ユニットが参照しなければならない記憶位置には入っていないためである。

余分な式の排除

余分な式とは，プログラムを正しく実行してゆく上で評価する必要のない式である。

この最適化の効果は，IFステートメントの中の論理式の使用を，一連のネストされたステートメント群よりも効率のよいものにすることである。たとえば，

```
IF (A=D)|(C=D) THEN X=Y+Z;
```

は，次のものより効率がよい。

```
IF A=D THEN X=Y+Z;
   ELSE IF C=D THEN X=Y+Z;
```

これは，AがDと等しければ，CとDが等しいかどうかをテストせずに，最初の例のTHENクローズが実行されるためである。

式の単純化

式の単純化とは，意図された効果を変えることなく，原始ステートメント式の形式を変更して，それらをより効率のよい目的コードにコンパイルすることができるようにする処理について述べるのに用いられることばである。

このコンパイラーでは，2つの形式の式の単純化が行なわれる。どちらの場合も，演算式の中で算術定数が用いられる。式の単純化は次のようにして行なわれる。3 * Bというような式は，B+B+Bと変形され，また，添字式の場合は，I+2というような式は，I * MULT+2 * MULTと変形される（MULTは定乗数）。このあと，2 * MULTは，アドレス計算のときオフセット因数として用いられる。

最適化コンパイラーのための原始プログラムのコーディング

このセクションでは，最適化機能を利用するために守らなければならないことまたは避けなければならないことに関連する実際のコーディングの詳細について述べる。

共通式の排除

次のような場合には，共通式の排除は，禁止される。

1. 入出力または計算のオン・ユニットの中でリセットされる可能性のある値をもつ変数の式の中で用いられているとき。

2. 基底付変数が，プログラム内のどこかで，共通式の中で用いられている変数の先頭からオーバーレーされている場合は，2つの共通式の間で基底付変数に新しい値を割り当てると，最適化は禁止される。たとえば，次の例の共通式X+Zの場合は，プログラムの前の方で基底付変数Aが変数Xにオーバーレーされており，2つのX+Zの間でAに値が割り当てられているので，共通式X+Zは排除されない。

```
DCL A BASED(P);
P = ADDR(X);
      .
      .
      .
P = ADDR(Y);
B = X+Z;
P->A = 2;
C = X+Z;
```

3. ORDERが指定されているときは，別名をもつ変数についての次の定義に該当している変数の場合。別名をもつ変数とは，それ自身の識別名以外の識別名を参照することによって修正される可能性のある変数，たとえば，DEFINED 属性をもつ変数，定義づけされた変数に対する基底として用いられている変数，パラメーター，ADDR組込み関数に対する引数，基底付変数などである。

外部にあるポインターまたは引数として用いられているポインターによって，外部プロシージャーにそのアドレスが知ら

れている変数も，別名をもつ変数とみなされる。

別名をもつ変数の効果は，共通式の排除を完全に阻止することではなく，一時的に禁止することである。別名をもつ変数のすべてに対して，コンパイラーは，別名をもつ変数を参照する可能性のあるすべての変数のリストを作成する。このリストは，リストのどのメンバーについても同じであり，ある1つのプログラムの中に，このようなリストがいくつもある場合もある。

別名をもつ変数が含まれている式が，共通式として使用されるかどうかチェックされるときは，その式の中で参照されている変数ばかりでなく，その変数が属している別名リストのすべてのメンバーについて，割当てが行なわれるかどうか，関連する共通式が出てくる可能性のある流れの道筋が探索される。プログラムに外部ポインター変数が含まれている場合は，外部プロシージャーに知られているアドレスをもつすべての変数，すなわち，すべての外部変数，外部プロシージャーに渡されるすべての引数，および外部ポインターに割り当てることのできるアドレスをもつすべての変数，に対して，このポインターをセットすることができるとみなされる。したがって，外部ポインターまたは外部ポインターからそれに割り当てられる値をもつ他のポインターは，その外部変数と同じ別名リストに属するものとみなされる。

4. 式の形式。B＋Cという式を共通式として取り扱うことができる場合でも，コンパイラーは，次のような式の中では，それを共通式として見つけ出すことはできない。

D ＝ A+B+C；

コンパイラーは，A＋B＋Cという式を左から右へ処理する。したがって，コンパイラーは，A＋Bという式と（A+B）＋Cという式を認識するだけである。しかし，D= A+（B＋C）という形で式をコーディングした場合は，コンパイラーは，優先順位の最も高い式を最初に処理しなければならないので，プログラマーは，それが認識されることを確信することができる。

5. 共通式の有効範囲。共通式の存在を判定するために，プログラムが分析され，流れの単位の存在が判定される。流れの単位とは，最初の命令のところで入り，最後の命令のところから出てゆくことだけが可能なブロック全体または一部分を表わす，コンパイルされたコードの一単位である。共通式は，個々の流れの単位を越えて認識される。しかし，いくつかの流れの単位の間にあるプログラムの流れ径路が複雑な場合は，流れの単位を越えての共通式の認識は禁止される。

共通式の排除は，以下の点によって助けられる。

1. 式の中の変数は，外部のものであったり，外部ポインターと結びつけられるものであってはならず，また，ADDR組込み関数の引数であってはならない。

2. 原始プログラムには，外部プロシージャー，外部ラベル変数，または外部プロシージャーに知られているラベル定数が含まれていてはならない。できる限り，ラベル・リストは，ラベル変数の宣言の中で与えなければならない。

3. 式の中の変数は，できる限り，オン・ユニットの中でセットしたり，アクセスしたりしてはならない。

不変の式の移動

次の場合には，ループから不変の式を移動することは禁止される。

1. そのブロックに対してORDERが指定されているとき，しかし，ORDERオプションによって移動が完全に禁止されるわけではなく，計算割込みが起こる可能性のある演算に対して禁止されるだけである。添字が2進ハーフワード整数によって表わされる場合は，このような演算には配列の添字の処理は含まれない。このような添字は，初期値化されないことがない限り（その場合には，いずれにしてもプログラムはエラーになるが），オーバーフローを起すことはない。

2. 入力または出力ステートメントによって，セットされたり，使用される可能性のある値をもつ変数を使用しているとき。

3. 入力，出力または計算オン・ユニットの中でセットされる可能性のある値をもつ変数，または別名をもつ変数を式の中で使用しているとき。

4. 外部プロシージャー，外部ラベル変数およびラベル定数を含んでいて，プログラムの流れが複雑なとき，およびラベル変数の宣言の中にラベル・リストがないとき。

移動は，次のものによって助けられる。

1. ブロックに対してREORDERを指定。

2. 上記の 2 − 4の点がとり除かれている。

余分な式の排除

余分な式の排除は，上述の不変の式の移動の場合と同じ要素によって，禁止されたり，助けられたりする。

その他の最適化機能

次のような項目について，最適化されたコードを生成することができる。

1. DOグループの制御変数に対して（DOループの実行中に明示的，またはオン・ユニットのどちらかによってその値が修正される可能性がある場合をのぞく）。

2. 他のDOループを含まないDOグループに対して（制御変数の有効範囲が，そのDOループを含んでいるブロックを越えているときは，DOループの後でブロックの終りよりは前のところで，限定された値が与えられることが条件）。

3. 非連続の記憶域が使用されていない限り，配列または構造の割当てに対して。

4. 配列が非連続の記憶域に入っているのでない限り，その各要素に同じ値が割り当てられるような配列の初期値化に対して。

5. 複雑なピクチャーや，文字から算術値への変換が含まれていない限り，インラインの変換に対して。

6. STRINGSIZEまたはSTRINGRANGEのON条件が解禁されていない限り，ストリング組込み関数SUBSTRおよびINDEXに対するインライン・コードに対して。

7. レジスターの割振りおよびアドレス体系に対して（外部プロシージャー，外部ラベル変数，外部プロシージャーに知られているラベル定数を使用しているためにプログラムの流れが複雑になっている場合，またはラベル変数の宣言の中でラベル・リストが与えられていない場合をのぞく）。別名をもつ変数や，オン・ユニットの中で参照されたり，セットされたりする変数を使用しているときも，レジスターの用法の最適化は禁止される。

プログラミングの手法

PL／Iでは，ある1つの効果を得るための方法がいくつもある場合が多くある。それらの方法のうちのあるものは，特定の観点からみて，他の方法にくらべて効率がよいということになるが，これは，関連する言語機能の実施の仕方によって大きく左右される。しかし，プログラムの効率の良否の原因となるものは，実際のプログラミングが始まる前の，問題定義の段階ですでに発生していることを，最初によく認識しなければならない。PL／Iの機能をよく考えて問題を定義しなければ，PL／Iを充分活用することはできない。

このセクションは，プログラマーがこのコンパイラーをうまく活用できるよう，その一助として書かれている。初めの2つの部分は，次のような2つの異なる観点から表わされている。

コンパイレーション速度の向上

実行の速度の向上

このセクションの残りの部分は，共通の関心事であり，記憶域の使用法，コンパイル時の機能の用い方，入出力機能の用法や，その他のヒントについて取り扱う。

コンパイレーションの速度の向上

NOOPTIMIZEコンパイラー・オプションは，できる限りコンパイレーションを速くするよう要求するものである。さらに，次のような方法も使用するとよい。

1. できるだけ多くの記憶域をコンパイラーに割り振る。このようにすると，コンパイラーが補助記憶装置にとり出すために費やされる時間が少なくなる。

2. 開始ブロックおよびプロシージャーの数を最小に抑える。ステートメントをグループ化するだけのためにBEGIN-ENDを使用しないこと。DOグループを使用すれば，より簡単に同じ効果が得られる。

3. リスティング，たとえば目的リスティングなどを作成するコンパイラー・オプションを用いることは，極力避けること。

4. 再実行の場合は，次のようにすると，わずかではあるが効率が上がる。

   a. 参照されないラベルとデータをすべて取り除く。

   b. 原始プログラムのエラーをすべて訂正し，さらに可能なところでは，"FILE／STRING option missing in GET／PUT statement" などのメッセージも含めて，診断メッセージの原因をとり除く。

5. プログラム・チェックアウトの補助手段を不必要に使うこと－－とくにCHECKステートメント－－を避ける。

実行速度の向上

OPTIMIZE=TIME コンパイラー・オプションは，プログラムの実行をできる限り速くすることを要求するものである。さらにまた，次のような方法も使用するとよい。

1. 不必要にプログラムをセグメント化したり，ブロック構造にするのを避ける。プロシージャー，オン・ユニット，および開始ブロックはすべて，プロローグとエピローグを必要とするので，相当多くのオーバーヘッドが含まれることになる。プログラムの論理を制御するためにGO TO またはIF ステートメントを使用すると，必要なGO TO または IFステートメントの数が多すぎない限り，CALLステートメントを使用するよりは，効率がよい。

2. 次の方法を，ラベルに対して使用することができる。

   a. GO TOステートメントは，ブロックの外側のラベルよりも同じブロック内のラベルを参照した場合の方が効率がよい。

   b. ラベル変数を参照するGO TO ステートメントは，その変数が表わすことのできるラベル定数の完全なリストがラベル変数の宣言に含まれていると，効率がよくなる。

3. 調節可能な配列および／またはCONTROLLED 記憶域の乱用を避けること。

4. 可能なところでは，式のかわりに定数を用いること。

5. 精度の指定に注意すること。たとえば，

```
DCL A FIXED DEC(8,4),
    B FIXED DEC(10,2),
    C FIXED DEC(10,1);
    .
    .
    .
C=A+B;
```

   この例では，Bを（10，4）と宣言した場合にくらべて約2倍のコードが必要になる（A+Bの評価に，スケール因数4が必要なため）。

6. 必要なときだけPICTURE 属性を使用すること。たとえば，PIC '999V99'のかわりにFIXED DECIMAL（5，2）を使用する。2つ以上の算術演算で同じピクチャー・フィールドが用いられている場合は，それを一旦変換してから，各演算では新しい形式を使用すること。これは，2回以上必要な変換のためにとっておかれる。

   算術式の中でPICTURE 属性をもつデータを使用する必要がある場合は，インラインで処理されるピクチャーを使用すること（このようにすると，実行時間が大巾に減るので）。全桁が9と1個のVをもち，浮動符号をもたないピクチャーは，とくに有効である。たとえば，

```
'999'
'$99V99'
'S99'
'V999'
```

7. 内部スイッチやカウンター，および実質的な計算に含まれるデータや添字に用いられるデータは，BINARY と宣言する方がよい。出力に必要なデータは，DECIMAL 形式で保存するとよい。

8. データ変換は，最少にとゞめること。そのためのいくつかの方法を以下に示す。

   a. 変換を追加して用いる。たとえば，文字変数を定期的に1ずつ増やしてゆかなければならないことを問題が指定している場合に，

```
DCL CTLNO CHAR(8);
.
.
.
CTLNO = CTLNO+1;
```

   上の例では変換が2回必要であるのに対して，

```
DCL CTLNO CHAR(8),
    DCTLNO DEC FIXED;
.
.
.
DCTLNO=DCTLNO+1;
CTLNO=DCTLNO;
```

   このようにすれば，変換は1度だけでよい。

   b. ある構造から別の構造にデータを移動したい場合は，2つの構造が一致するようとくに注意しなければならない。

   c. 混合モードの算術演算は避けること。とくに，算術演算の中で文字ストリングを使用するのは避ける。

9. データの集合体に対して，次の方法を用いる。

   a. 一般に，配列式は反復DOグループに拡張し，構造式は一連の要素式に拡張する。しかし，次のような形式の単純な割当ての場合は，

```
A=B;
```

AとBが配列または構造ならば，Bは A全体に1まとまりで複写される。AとBは，調節可能な限界，長さ，またはサイズをもつものであってはならず，その参照は，連結された記憶域に対するものでなければならない。

b. 構造の中で，余分なレベルを宣言することを避ける。たとえば，

```
DCL  1 A,
       2 B,
        3 C(10),
        3 D CHAR(20);
```

この例では，Bは余分である。

c. ストリーム方式入力の中で，集合体の引数をもつ擬似変数を参照することを避ける。

d. 集合体の式の中で，ユーザーの関数や配列取扱い組込み関数を参照することを避ける。

e. 関数，とくに集合体の引数をもつ関数に対する，ネストされた参照を避ける。

f. 集合体の宣言は，引数としてそれらを渡さずに，できる限り，それらが使用されているプロシージャーの中で行なうこと。1つの集合体が2つ以上のプロシージャーの中で必要な場合は，そのプロシージャーが外部のものであればEXTERNAL と宣言し，外部でなければそれが使用されている一番外側のブロックの中で宣言しなければならない。

g. 添字変数は，それらがFIXED BINARY (15,0) として用いられているブロックの中で宣言すること。

10. ストリングについては，次のような方法を用いることができる。

a. ビット・ストリングは，できるだけ，8ビットの倍数として指定すべきである。しかし，論理スイッチとして用いられるビット・ストリングは，必要なスイッチの数に合わせて指定しなければならない。以下の例では，(a)は(b)より望ましく，(b)は(c)よりは望ましい。

例1：

単一スイッチ

```
(a)  DCL SW BIT(1) INIT('1'B);
      .
      .
      .
     IF SW THEN DO;
      .
      .
      .
(b)  DCL SW BIT(8) INIT('1'B);
      .
      .
      .
     IF SW THEN DO;
      .
      .
      .
(c)  DCL SW BIT(8) INIT ('1'B);
      .
      .
      .
     IF SW = '10000000'B THEN
     DO;
      .
      .
      .
```

例2；

複数スイッチ

```
(a)  DCL B BIT(8);
      .
      .
      .
     B = '11100000'B;
      .
      .
      .
     IF B ='11100000'B THEN DO;
      .
      .
      .
(b)  DCL B BIT(3);
      .
      .
      .
     B = '111'B;
      .
      .
      .
     IF B = '111'B THEN DO;
      .
      .
      .
(c)  DCL (SW1, SW2, SW3) BIT (1);
      .
      .
      .
     SW1, SW2, SW3, = '1'B;
      .
      .
      .
     IF SW1&SW2&SW3 THEN DO;
      .
      .
      .
```

長さが8の倍数でないビット・ストリング・データが構造の中にある場合は，このような構造はALIGNED と宣言しなければならない。

b. ビット・ストリングに対する連結演算は，時間がかかることに注意していただきたい。

c. 長さの変わるストリングは，一般に，固定長のストリングにくらべて効率が悪い。

d. 固定長ストリングは，次の例に示すように，コンパイル時にその長さが知られていないと，効率が悪い。

```
DCL A CHAR(N);
```

11. デバッギング中以外は，SIZE，SUBSCRIPTRANGE，STRINGRANGE，およびCHECK ON-条件を使用するのは避けること。デバッギングの補助手段は，生産的なジョブとしてプログラムを操作する前に，とり除かなければならない。

12. 1回の操作で，2度以上DATE組込み関数を参照しないこと。かわりに，関数を1回参照して，その値を後で使用するために変数の中に保存しておく。たとえば，次のようにするかわりに，

```
PAGEA= TITLEA||DATE;
PAGEB= TITLEB||DATE;
```

次のように書く方が効率がよい。

```
DTE=DATE;
PAGEA=TITLEA||DTE;
PAGEB=TITLEB||DTE;
```

13. 入出力に対して，次のような方法を用いることができる。

a. プログラムが，入出力に制約されることがないように，緩衝域を充分に割り振ること。

b. ブロック化された出力レコードを使用すること。

c. 1つのOPENステートメントで，いくつかのファイルを開くこと。

d. STREAM入出力では，入出力ステートメントを分割するかわりに，長いデータ・リストを使用すること。

e. リスト指示またはデータ指示よりはむしろ，編集指示入出力を使用する方がよい。

f. 文字ストリング構造との間での転送を単純にするために，オーバーレー定義を使用することを考えてみること。たとえば，

```
DCL 1 IN,
      2 TYPE CHAR(2),
      2 REC,
         3 A CHAR(5),
         3 B CHAR(7),
         3 C CHAR(66);
GET EDIT(IN)
  (A(2),A(5),A(7),A(66));
```

上の例では，形式項目／データ・フィールドの各対は別々に突き合わせられ，突合せ操作の各々に対してコードが生成される。構造の上に文字ストリングを定義し，そのストリングに対してGETステートメントを用いると，効率がよくなる。

```
DCL STRNG CHAR(80) DEF IN;
 .
 .
 .
GET EDIT (STRNG) (A(80));
```

g. ファイルがDIRECT INDEXEDと宣言されている場合は，INDEXAREA，NOWRITE，およびADDBUFなどのENVIRONMENTオプションを，できるだけ用いるとよい。

h. CONSECUTIVEデータ・セットの作成またはアクセスをするときは，インライン・コードが生成されるように，できる限り，ファイルおよびレコード変数の宣言を使用すること。宣言についての詳細は，第12章「レコード方式の転送」のところで説明されている。

i. REGIONAL データ・セットに対する原始キーの変換は，次のような特別の場合には，排除することができる。

(1) REGIONAL(1)の場合： 原始キーが，固定2進数の要素変数，または範囲が(12,0)から(23,0)までの精度をもつ定数のとき。

(2) REGIONAL(3)の場合： 原始キーが，（文字ストリング式||r）という形式で，しかも"r"が固定2進数の要素変数，または範囲が（12,0)から(23,0)までの精度をもつ定数のとき。

j. 同じキーに対するREADとREWRITEの間に，同じファイルに対する入出力操作が介在している場合は，INDEXEDデータ・セットの直接的な更新の速度が落ちる。このような場合は，REWRITEステートメントが余分なREADを出す結果になることもある。

（入出力については，この章の「入出力機能の用法」のところでも説明する。）

14. 言語間の連絡に，次の方法を用いることができる。

a. 可能な限り，PL／Iの集合体引数が，COBOLまたはFORTRANの集合体と同じようにマップされているようにする。

b. PL／IとCOBOLの構造が同じようにマップされるときを，コンパイラーがつねに発見できるわけではない。基底の構造の中の各要素は，境界合せの必要条件をもっている。たとえば，CHAR(4)項目は，主記憶域の中でどのバイトにでも境界合せすることができるが，それに対して，FIXED BIN(31)項目は，フルワードに合わせなければならない。構造の最初の基底要素の境界合せの条件が，他の基底要素にくらべて厳重でない場合はつねに，コンパイラーは構造に対して見かけ引数を作成する。この規則は，レベル2の各小構造に対しても，独立に適用される。項目の実際の長さが，挿入されるパディング用バイトを必要としないときは，NOMAP オプションを指定しなければならない。たとえば，

```
DCL 1 S,
      2 X CHAR(4),
      2 Y FIXED BIN(31);
```

Yの境界合せの条件が，Xよりも厳密であるため，コンパイラーは，この構造に対して見かけ引数を作成する。しかし，この場合パディングは不要なので，この構造はPL／IとCOBOLでは全く同じにマップされる。

c. 引数のマップが異なっている場合は，見かけ引数を不必要に初期値化するのを避けるためにNOMAPIN オプションを使用し，さらに，最終値が不要ならば（たとえば，値が変わらないとき），見かけ引数からの不要な割当てを避けるために，NOMAPOUT オプションを使用すること。

d. 次のようにすることによって，PL／I環境の初期設定が何度も行われるのを避ける。

(1) 主プロシージャーをPL／Iにする，または

(2) PL／Iプロシージャーが主ルーチンから呼び出されるようにする。または

(3) いくつかのPL／Iプロシージャーを呼び出している間に，PL／I環境がこわされないような，プログラム構造にする。

15. プログラム論理の一部分としてオン・ユニットを使用するのを避ける－－オン・ユニットが実行されると，オーバーヘッドが著しくとられる。たとえば，キーをもつデータ・セットに対してレコード方式の出力を使用するプログラムの中で，ある特定のキーは，データ・セットの範囲に収まらないため排除したい場合がある。このようなときに，不適当なキーをみつけるためにKEY条件を引き起す方法を使うこともできるが，各々のキーを明示的にテストする方が効率がよい。

インライン操作

多くの操作は，インラインで処理される。したがって，どの操作がインラインで行なわれるのか，またどれがライブラリー呼出しを必要とするのかを認識して，できれば前者が使用されるようプログラムを編成すれば，ユーザーはよい効果を上げることができる。
このようなインライン操作の大多数は，データ変換とストリングの取扱いに関するものである。

データ変換

インラインで行なわれるデータ変換が，図16.1に示してある。ある変換がインラインで行なわれるかどうかは，最小の実行時間が要求されているかどうかによって決まる。「最適化」の欄には，要求されている最適化のレベルが重要でないときは「なし」と記入されており，実行時間の最適化が必要なときは「時間」と記入されている。与えられた範囲または条件の外にある変換，または「行なわれない」と記入されている変換は，ライブラリー呼出しによって実行される。

使用可能なピクチャー文字のすべてが，インラインの算術変換に関係するピクチャーの中で使用できるとは限らない。使用できるのは，次のものだけである。

Vと9

浮動，または非浮動文字　$　S　+

ゼロ消去文字　Z　*

句読点文字　,　.　／　B

インラインの変換の場合，この文字のサブセットからなるピクチャーは，3つのタイプに分けられる。

ピクチャー・タイプ1：　1個のV（任意選択）と先頭または末尾に符号をもつ，全桁が9のピクチャー。たとえば，

変換 原フィールド	変換 目標フィールド	注および条件	最適化 SIZE（禁止）	最適化 SIZE（解禁）
FIXED BINARY	FIXED BINARY	−	なし	なし
	FIXED DECIMAL	どちらかのスケール因数＝0で，他方のスケール因数≤0のときは，最適化は「なし」になることがある。	時間	時間
	FLOAT	原フィールドのスケール因数＝0のときは，最適化は「なし」になることがある（SIZEが解禁されているかどうかによって）	時間	時間
	ビット・ストリング	ストリングは，固定長，ALIGNEDで，長さ≤2048のものでなければならない。	なし	行なわれない
	文字ストリングまたはピクチャー	原フィールドのスケール因数は≥0でなければならない。ストリングは，長さ≤256でピクチャー・タイプ1,2,または3の固定長のものでなければならない。	時間	行なわれない
FIXED DECIMAL	FIXED BINARY	原および目標フィールドのスケールが同じ符号で，ゼロでないときは，最適化（SIZE禁止）は「時間」でなければならない。	なし	時間
	FIXED DECIMAL	−	なし	なし
	FLOAT	原フィールドの精度は≤10でなければならない。	時間	時間
	ビット・ストリング	原フィールドのスケール因数はゼロでなければならない。ストリングは，固定長，ALIGNEDで，長さが≤2048でなければならない。	なし	行なわれない
	文字ストリング	原フィールドのスケール因数は≥0でなければならない。ストリングは，固定長で長さが≤256でなければならない。	時間	時間
	ピクチャー	ピクチャー・タイプ1，2，および3。符号なしのピクチャー・タイプ1と2の場合は，最適化は「なし」になることもある。	時間	行なわれない
FLOAT	FIXED BINARY	−	時間	行なわれない
	FIXED DECIMAL	目標フィールドの精度は，≤9でなければならない。	時間	行なわれない
	FLOAT	原および目標フィールドの長さは，単でも倍でもよい。	なし	なし
	ビット・ストリング	ストリングは，固定長，ALIGNEDで長さ≤2048でなければならない。	時間	行なわれない
ビット・ストリング	FIXED BINARY	原フィールドのストリングは，固定長，ALIGNEDで，長さ≤2048でなければならない。	なし	行なわれない
	FIXED DECIMAL および FLOAT	原フィールドは，固定長，ALIGNEDで，長さ<32でなければならない。	時間	行なわれない

図16.1（2の1）　インラインで行なわれる暗示的なデータ変換

変換		注および条件	最適化	
原フィールド	目標フィールド		SIZE（禁止）	SIZE（解禁）
ピクチャー	文字ストリング	ストリングは，長さ≦256の固定長のものでなければならない。	なし	なし
	ピクチャー	両方のピクチャーが同じでなければならない。	なし	なし
ピクチャー・タイプ1	FIXED BINARY	原フィールドの精度は，＜10でなければならない。	時間	行なわれない
	FIXED DECIMAL	ピクチャーが符号をもっているときは，最適化は「時間」でなければならない。	なし	行なわれない
	FLOAT	原フィールドの精度は，＜10でなければならない。	時間	行なわれない
	ピクチャー	ピクチャー・タイプ1，2，または3	時間	行なわれない
ラベル	ラベル	−	なし	なし
ロケーター	ロケーター	−	なし	なし

図16.1（2の2） インラインで行なわれる暗示的なデータ変換

'99V999'，'99'，'S99V9'，'99V+'，'$999'

ピクチャー・タイプ2： ゼロ消去文字，および（任意選択で）句読点文字，および符号文字をもつピクチャー。句読点文字をもつタイプ1のピクチャーもこれに当たる。たとえば，

'ZZZ'，'**/**9'，'ZZ9V.99'，

'+ZZ.ZZZ'，'$///99'，'9.9'

ピクチャー・タイプ3： 浮動ストリング，および（任意選択の）挿入文字，および符号文字をもつピクチャー。たとえば，

'$$$$'，'-,--9'，'S/SS/S9'，

'+++9V.9'，'$$$9-'

ピクチャーが上記のタイプのいずれかであっても，ピクチャー変換が行なわれない場合が，ときにはある。これは，次のような理由のためである。

1. 最適化のレベルが低すぎる。
2. SIZEが解禁されている。
3. 原フィールドと目標フィールドの数字桁の間にオーバーラップがない。たとえば，

   DECIMAL(6,8)またはDECIMAL(5,−3)からPIC '999V99'へは，実行されない。

4. ピクチャーが，インラインで処理するのをむずかしくするある種の性質をもっている。たとえば，

   a. 浮動Zまたは浮動*と，最初の9の間の句読点の前にVがない。たとえば，

      'ZZ.99'

   b. 小数点の右側に，浮動文字またはゼロ消去文字がある。たとえば，

      'ZZV.ZZ'，'++V++'

ストリングの取扱い

インラインで行なわれるストリング関数と演算が図16.2および図16.3に示されている。つねにインラインで行なわれると表示されているストリング関数が，ライブラリー・ルーチンを呼び出すこともあるので注意しなければならない。たとえば，BITまたはCHAR関数の中の式が，インラインで処理されない暗示的な変換を必要とする場合は，該当するライブラリー・ルーチンが呼び出される。

記憶域の用法

記憶域スペースを減らすことの方が，実行時間を減らすことよりも重要な場合は，次のような方法を用いることができる。

ストリング演算	注 および条件		
	原フィールド	目標フィールド	注
割当て	調節不可能，ALIGNED，固定長のビット・ストリング	調節不可能，ALIGNEDのビット・ストリング	OPTIMIZE=TIMEが指定されている場合は，長さの制約はない。それ以外の場合は，最高8192ビット。
	調節不可能またはVARYING，ALIGNEDのビット・ストリング。	調節不可能，ALIGNEDで，≦2048 ビットのビット・ストリング。	OPTIMIZE=TIMEが指定されているときだけ。
	調節可能な限界または範囲をもたないAUTOMATIC，BASED，またはSTATIC 構造の一要素である，調節不可能，UNALIGNED，固定長のビット・ストリング。	(原フィールドと同じ)	OPTIMIZE=TIMEが指定されているときだけ，最大の長さ=57ビット。
	調節不可能，固定長の文字ストリング。	調節不可能の文字ストリング。	
	調節可能またはVARYING の文字ストリング。	調節不可能で，≦256の文字ストリング。	
'and'，'not'，'or'	ビット・ストリング割当ての場合，しかし，調節可能または長さの変わるオペランドは，処理されない。		
比 較	原フィールドと目標フィールドの役割をする2つの比較オペランドによる割当ての場合。しかし，調節可能または長さの変わるオペランドは，処理されない。		
連 絡	ストリング割当ての場合，しかし，調節可能または長さの変わる原ストリングは，処理されない。		
STRING 関数	連続記憶域にある，要素変数，および調節不可能な配列および構造変数。		

注： 1. 指定されている最大の長さは，オペランドではなく演算の長さを指している。目標フィールドが固定長のときは，演算の長さは目標フィールドの長さになる。目標フィールドがVARYING のときは，演算の長さは，両オペランドの長さのうち短い方になる。

2. パラメーター，定義づけされた変数，または集合体変数の一部分であるUNALIGNED ビット・ストリングは，処理されない。

図16.2 ストリング演算がインラインで処理される場合の条件

1. 計算に用いられる固定小数点データは，できる限り，10進数ではなく，2進数にすべきである。

2. INDEXED として宣言されているファイルが，DIRECT UPDATE として用いられているが，それにレコードが追加されることはないという場合は，ENVIRONMENT オプションのNOWRITE を使用すると，データ管理プログラムに関して約5000バイトの記憶域が節減される。

3. 境界合せの属性：これらの属性を用いることによって，ユーザーは，算術およびストリング・データに対して，次のように境界合せを行なうことができる。

ALIGNED：

算術：

FIXED DECIMAL： バイト

ストリング関数	注 および条件
BIT	つねに
BOOL	3番目の引数は，定数でなければならない。最初の2つの引数は，図16.2に示される 'and'，'or'，および 'not' 演算に対する条件を満足していなければならない。
CHAR	つねに
HIGH	つねに
INDEX	2番目の引数は，長さ<256文字の，調節不可能な文字ストリングでなければならない。
LENGTH	つねに
LOW	つねに
REPEAT	2番目の引数は，定数でなければならない。
SUBSTR	STRINGRANGEは，禁止しなければならない。
TRANSLATE	最初の引数は固定長でなければならず，2番目と3番目の引数は定数でなければならない。
UNSPEC	つねに
VERIFY	最初の引数は固定長でなければならない。CHARACTER のときは≦256文字でなければならず，BITのときは ALIGNED で，≦2048ビットでなければならない。2番目の引数は定数でなければならない。

図16.3　ストリング関数がインラインで処理される場合の条件

FLOAT(DOUBLE)：　ダブルワード
FIXED BINARY
(p,q)：　ハーフワード
(ただしP≦15)
その他：　ワード

ストリング：　バイト

UNALIGNED:

算術ストリング：　バイト
文字ストリング：　バイト
ビット・ストリング：　ビット

したがって，パディングを最少にして，主記憶域内における データのパック化の密度を濃くするために，UNALIGNED 属性を使用することができる。

エリアおよび事象データは，つねにワードまたはダブルワード境界に合わせられる。これらをUNALIGNEDにすることはできない。

データの集合体の中で，集合体に対して明示宣言された境界合せは，その集合体の各要素に適用される。しかし，構造の場合，この境界合せは，個々の基底要素に対して指定された境界合せによって置き換えることができる。たとえば，

```
DCL 1 STR UNALIGNED,
      2 A,
      2 B ALIGNED,
      2 C;
```

この場合，AとCはUNALIGNED，BはALIGNEDになる。

省略時解釈の属性は，データ項目の場合も，データ集合体の場合も，関係する要素のデータ・タイプによって決まる。これらの省略時解釈は次の通りである。

UNALIGNED　すべてのストリング・データと PICTURE 項目

ALIGNED　次のような，すべての算術データ

BINARY
DECIMAL
FIXED
FLOAT

たとえば，

```
DCL A BIT(4),
I,
(B CHAR(10),X) UNALIGNED,
(C BIT(12),Y FIXED) ALIGNED;
```

この場合，Aは省略時解釈によりUNALIGNED，Iは省略時解釈によりALIGNEDになるが，B，C，X，およびYは，明示宣言された通りになる。

```
DCL (A1(80) CHAR(6), A2(80) BINARY)
     ALIGNED,
    B1 (3,3) BIT(2),
    C1 (3,3) CHAR(4),
    D1 (100) DECIMAL;
```

この場合，A1とA2は，明示宣言された通りになり，B1とC1は省略時解釈によってUNALIGNED，D1は省略時解釈によりALIGNEDになる。

```
DCL 1A,
      2 B,
      2 C BIT(4) UNALIGNED,
      2 D ALIGNED,
        3 E BIT(2),
        3 F,
      2 G CHAR(10);
```

ここで，Aは大構造，
Bは省略時解釈により，FLOAT DECIMAL ALIGNED,
Cは明示宣言によりUNALIGNED,

Dは小構造,
EはBIT ALIGNED(Dから受けつぐ),
Fは省略時解釈によりFLOAT DECIMAL
ALIGNED(この場合, ALIGNED はDから受けついだものであるが, DがALIGNED と宣言されていない場合でも, これがFに対する省略時解釈である)。
GはCHAR UNALIGNEDである。

以下の点に関して変数の突合せをするとき, ユーザーは, 境界合せの属性が正しいかどうか注意しなければならない。

a. DEFINED属性の使用

b. 引数および関連するパラメーター

4. 次の例で示すようなDOグループは, 時間の面からは効率がよいが, 記憶域の面からみると無駄が多い。

```
DO  I  =  1,2,6,9;
  .
  .
  .
  END;
```

記憶域を節減するためのもう1つの方法は, 次の通りである。

```
DCL VALUES(4) FIXED BIN STATIC
INIT(1,2,6,9);

DO  J=1  TO  4;
I=VALUES(J);
  .
  .
  .
END;
```

5. デバッギングが終ったら, デバッギングの目的で解禁されていた条件(普通は禁止されるもの)は, NO-条件接頭語をつけるのではなく, 関連する接頭語をとり除くことによって禁止すること。たとえば, SIZE条件の場合は,NOSIZE接頭語をつけるのではなく, SIZE接頭語をとり除くことによって禁止する。後者の方法を用いた場合は, コンパイラーは, その条件をチェックするコードを除去することができるが, 前者の方法では, チェックが行なわれないようにするための余分なコードを生成する必要がある。

入出力機能の用法

大量の入出力を行なうプログラムでは, データ・セットの性質如何によって(レコード形式, 長さ, ブロック化因数, など), 時間のオーバーヘッドが大巾に違ってくる。一般に, レコードをブロック化すると, 時間とデータ・セット上のスペースの両方が節減される。

ファイルを開く作業を別々に行なうと(明示的または暗示的のどちらであっても), 常駐ライブラリーからOPENモジュールをロードし直す必要があるので, 1つのOPENステートメントでいくつものファイルを開く方が有利である。ファイルが開かれるとき, 緩衝域または作業スペースと, データ管理プログラムとのインターフェース・モジュールが記憶域に入るということも考慮に入れなければならない。したがって, 記憶域スペースを第一に考えなければならないときは, OPEN およびCLOSEステートメントを充分考慮して使用すると, 使用可能な記憶域をうまく制御することができる。

STREAM 転送に関する3つの手法のうち, データ指示転送は, 通常, 時間もスペースも最も多く必要である(転送されるデータ変数の各々に対して, 記号テーブル記入項目が目的時に存在している)。リスト指示は, フリー形式の入出力に使用することができるが, 編集指示を使用することによって最も高度の制御を得ることができる(編集指示は, 通常, 目的時におけるスペースと実行時間の点で最も効率のよい手法である)。ストリーム入出力操作を繰返し実行するための最もよい方法は, DOループの中にステートメントを入れる方法ではなく, 入出力ステートメントの中に繰返しDO指定を入れる方法である。繰返し指定を用いると, 緩衝域を設定するなどの初期的なハウスキーピング操作を行なうためのライブラリー呼出しを繰り返す必要がなくなる。

RECORD 転送は, データの集合体を, 要素から要素へという形ではなく, データ全体を1つのものとして取り扱う機能をもっている。したがって, データの変換も編集も行なわれないので, データ・セット・レコードを効率よくアクセスするという点, およびデータ・セット・スペースを効率よく使用するという点で, 有利である。位置指定方式の転送を用いれば, 緩衝域と変数の間でのデータの移動が節減される。

INDEXED データ・セット編成を使用しているときに, あふれ域の中でレコードをアクセスすると, 処理速度が著しく遅くなる可能性がある。INDEXEDデータ・セットを定期的に作り直して, 論理的には削除されているが, 物理的には存在しているレコードをデータ・セットから追い出し, あふれ域にあるレコードを主要データ域に集めるようにするとよい。

順次アクセスで使用可能な, BUFFEREDとUNBUFFEREDファイル属性のいずれを選択するかについては, とくによく考慮する必要がある。BUFFEREDファイルの場合は, 目的プログラムは, 「予期的な緩衝法」, すなわち, 装置の転送と計算の時間のオーバーラップをすることが可能になる。UNBUFFEREDファイルの場合は, EVENT オプションが使用されていない限り, このようなオーバーラップをすることはできない。(UNBUFFEREDファイルは, つねに隠された緩衝域を必要とすることに注意していただきたい。)

入出力エラーの回復

入出力転送エラーに出合うと，自動的にデータ管理プログラムによって余すところなく回復処理手順が行なわれるので，TRANSMITオン・ユニットに入ってから，それ以上エラー訂正手順を実行する必要はないし，また不可能である。転送エラーと関連するオン・ユニットに入ることとが確実に同期できるのは，入力エラーの場合だけである。出力操作のエラーの場合は，該当するエラーは，ブロック化因数や割振られた緩衝域の数によって決まるので，後続の出力操作の中で発見されることもある。

その他のヒント

オペレーティング・システムとジョブ制御

内部プロシージャーとは異なり，外部プロシージャーは，別個の制御セクションとして取り扱われる。

宣言と属性

1. 省略時解釈にあまり頼りすぎないこと。明示的に宣言すれば，原始プログラムの論理は明確になるし，ある場合には（たとえば，精度）エラーの可能性も減る。

2. FIXED BINARY またはFLOAT BINARYと宣言された変数は，それらが単一のものであるか，集合体の一部であるかには関係なく，自動的に正しいワード境界に合わせられる（UNALIGNEDと宣言されていない限り）。FIXED DECIMAL 変数は，パック10進数形式で貯えられ，それらが関係する演算では，システム／360の10進数命令が用いられる。FLOAT DECIMAL 変数は，浮動小数点形式で貯えられる。それらが関係する演算は，浮動小数点命令セットを用いて実行される。

割当てと初期値化

1. 算術変数の割当てまたは初期値化で必要なときは，高位にゼロが挿入される。

```
DCL A FIXED DECIMAL (5,2) INIT (12);
/*A HAS VALUE 012.00*/
DCL B FIXED BINARY (15,0);
B=12;
/*B HAS VALUE 000000000001100B*/
```

2. 要素式から割り当てることによって，配列を初期値化することができる。

```
DCL A(10);
A=0;
```

配列の各要素に要素値が割り当てられる。同様に，要素式が構造に割り当てられるときは，構造の各要素にその値が割り当てられる。

```
DCL 1B,
      2 C BIT (1),
      2 D CHAR(1),
      2 E CHAR (4);
B=0;
```

この割当ての結果，それぞれの要素の値は次のようになる。

```
C '0'B
D 'b'
E 'bbb0'
```

2種類またはすべての3種類のデータ・タイプを含む構造の中で，すべての算術要素およびビット要素にゼロを割り当て，すべてのストリング要素にブランクを割り当てる必要があるときは，構造には，次のようなナル・ストリングが割り当てられることがある。

```
B='';
```

したがって，要素の値は次のようになる。

```
C   '0'B
D   'b'
E   'bbbb'
```

DO グループ

反復を後向きにすることもできるので，WHILE オプションの中の式で，制御変数を参照することができる。

```
DO I=N+2*L BY -X WHILE (I>0);
END;
```

このループの中で，制御変数を修正することができる。

DOループの中から，そのグループに対するENDステートメントのラベルに移行することができる。このようにすると，中間的な処理をせずに制御変数を増やす効果が得られ，制御は失敗しない。

また，制御変数の最終値に達する前に反復DOグループの外へ移行することもできる。

関　数

関数参照の中にある引数を，関数によって修正することができる。

オン条件とオン・ユニット

条件接頭語の有効範囲に注意していただきたい。

```
   (SIZE):A:PROC;
             .
             .
             .
 (NOSIZE):IF M>N THEN DO;
           J=E+F;
           END;
          .
          .
          .
          END A;
```

上の例では，M>N；という式を評価する間だけ，SIZEは禁止される。J＝E＋Fという割当てのときは，SIZEは解禁されている。

一般的なエラーと落し穴

次にあげるリストは，PL／Iの原始プログラムを書くときに陥りやすいエラーと落し穴を上げたものである。これらの項目のいくつかは，言語の規則を誤解したり，見逃したりしたためのものであるが，それ以外のものは，そのシステムでの約束事や制約を守らなかったために起るものである。

オペレーティング・システムとジョブ制御

オーバーレー・セグメントの中にあるSTATIC変数は，根セグメントの中にあるのでない限り，オーバーレー操作中に上書きされてもよい。

原始プログラムと一般的な統語規則

1. 次のような文字を書くときは，とくに注意しないと，穿孔の転記エラーが起る可能性がある。

   1（数字），　I（英字），|（または），
   ／（斜線），　'（引用符）；

   ¬（not），7（数字），
   >（より大）；

   L（英字），　<（より小）；

   O（英字），　0（ゼロ）；

   S（英字），　5（数字）；

   Z（英字），　2（数字）；

   –（分割文字），
   -（負符号）；

2. MARGINSオプションで指定されているマージンの中に，原始プログラムが完全に収まっているか確かめること。

3. ある種の記号を不注意でぬかしてしまうと，追跡が困難なエラーが起こることがある。このような一般的なエラーには，相手のない引用符，相手のない括弧，相手のない注釈区切り符号（たとえば，注釈の終りに*／のかわりに／*と穿孔した場合など），セミコロンがない，などの場合がある。

4. 48文字セットの場合の，予約キーワード演算子（たとえば，GT，CATなど）は，どんな場合も，その前後にブランクか注釈がなければならない。

5. ENDステートメントが，該当するDO，BEGIN，およびPROCEDUREステートメントと正しく一致しているかどうか，よく注意して確かめなければならない。

6. ある場合には，その必要性が直接的に明らかでないときでも，括弧が必要である。とくに，WHILEおよびRETURNの後にある式は，括弧で囲まなければならない。

プログラムの制御

1. 実行時に最初に制御が渡されるプロシージャーは，OPTIONS（MAIN）属性をもっていなければならない。2つ以上のプロシージャーがMAINオプションをもっているときは，連係編集プログラムが最初に出あった方が制御を得る。

2. プログラムのプロシージャーが，まだ活用状態にあるときに呼び出されたときは，回帰的に用いられているという。RECURSIVE属性が与えられていないプロシージャーを回帰的に使用しようとすると，そのプロシージャーから出たあとでプログラム割込みが起こる可能性がある。このようなこ

とは，そのプロシージャーが前に呼び出されたときの自動データを参照したときに起こる。

宣言と属性

1. AUTOMATIC 変数に対するDECLARE ステートメントは，実際にはブロックに入るときに実行される。次のタイプの一連のステートメントは，使用してはならない。

```
A:  PROC;
    N=4;
    DCL  B(N)  FIXED;
    .
    .
    .
    END;
```

2. DECLARE ステートメントの中でカンマをぬかすということは，原始プログラムに共通のエラーである。たとえば，構造宣言の中の各要素に対する記入項目の後にはカンマをつけなければならない。

3. 外部識別名は，7桁を越えてはならないし，またIBMという文字で始めてはならない。

4. PICTURE 宣言の中で，Vという文字は，スケール因数を表わしているが，それ自体出力の中で小数点にはならない。小数点ピクチャー文字は，出力の中で小数点になるが，これは純粋に編集文字であって，スケール因数を表わすものではない。しかし，10進数の定数では，小数点はスケール因数を表わしている。たとえば，

```
DCL A PIC'99.9',
      B PIC'99V9',
      C PIC'99.V9';
A,B,C=45.6;
PUT LIST (A,B,C);
```

上の例では，A，B，Cに対してそれぞれ次のような値が印刷される。

```
04.5    456     45.6
```

これらの値が，今度はGET LISTステートメントによって変数に読み戻されたとすれば，A，B，Cはそれぞれ次のような値にセットされる。

```
004     56.0    45.6
```

次に，PUTステートメントを繰り返したとすれば，その結果は次のようになる。

```
00.4    560     45.6
```

5. 同じ識別名に対する別の外部宣言では，明示的，省略時解釈のいずれによっても，矛盾する属性を指定してはならない。このようなことが起っても，コンパイラーはその矛盾を発見することはできない。

6. 1つの識別名を，その有効範囲内で2つ以上の目的に使用することはできない。したがって，次の一連のステートメントの中でXを使用すると，エラーになる。

```
PUT FILE (X) LIST (A,B,C); X=Y+Z;
X: M=N;
```

7. 10進整定数を渡すときは，そのような定数の精度も考慮しなければならない。たとえば，

```
CALL ALPHA(6);

ALPHA:  PROCEDURE(X);
        DCL X FIXED DECIMAL;
        END;
```

ALPHAが外部プロシージャーのときは，Xに省略時解釈の精度が与えられるので，上の例は正しくない。それに対して，定数，6は，精度（1，0）をつけて渡される。

8. あるデータ項目を見かけ引数に変換する必要があり，呼び出されたプロシージャーがパラメーターの値を変えるという場合は，見かけ引数は変えられるが，もとの引数は変えられないことに注意していただきたい。たとえば，

```
   DCL A FIXED,
       B FLOAT;
   CALL X(A,B);
 X:PROC(Y,Z);
   DCL (Y,Z) FIXED;
   Y=Z**100;  /*A IS ALTERED IN
                CALLING PROC*/
   Z=Y**3;    /*B IS UNALTERED IN
               CALLING PROC*/
   END X;
```

9. ある識別名に対する属性が全部宣言されていないときは，必要な残りの属性は，省略時解釈によって補なわれる。次のような省略時解釈の想定には，充分注意しなければならない。

FLOAT DECIMAL(6)REALは，暗示的に宣言された算術変数とみなされる。ただし，頭文字がIからNの範囲にあるものは例外で，この場合はFIXED BINARY（15,0）REALとみなされる。

変数が明示的に宣言されており，しかも基数，スケール，またはモードのいずれかが宣言されているときは，その他の属性は，FLOAT/DECIMAL/REAL のセットから想定される。たとえば，

```
DCL I;          /*I IS FIXED BINARY
                  (15,0) REAL
                  AUTOMATIC*/
DCL J REAL;     /*J IS FLOAT DECIMAL
                  (6) REAL
                  AUTOMATIC*/
DCL K STATIC;/*K IS FIXED BINARY
                  (15,0) REAL
                  STATIC*/
DCL L FIXED;  /*L IS FIXED DECIMAL
                  (5,0) REAL
                  AUTOMATIC*/
```

10. 複素数式の精度は，明らかでない。たとえば，1＋1Ｉの精度は(2,0)である。つまり，この精度は式の評価の規則にしたがう。

11. 1つのプロシージャーに2つ以上の入口点が含まれており，各々の入口ごとに別のパラメーター・リストがついているときは，そのプロシージャーに制御が入った地点で結びつけられたパラメーター以外のものは，参照しないようにしなければならない。たとえば，

```
A: PROCEDURE(P,Q);
   P=Q+8; RETURN;
B: ENTRY(R,S);
   R=P+S; /* THE REFERENCE TO P
             IS AN ERROR*/
   END;
```

12. 基底付記憶域は，ダブルワードの単位で割り振られる。したがって，最小の項目に対しても，少なくとも8バイトが必要である。

13. REFER オプションの中で用いられている変数は，不明確にならないように参照しなければならない。たとえば，

```
DCL 1 A,
      2 Y FIXED BIN,
      2 Z FLOAT,
    1 B,
      2 Y FIXED BIN,
      2 T(1:N REFER(B.Y));
```

この宣言を参照するときはつねに，不明確にならないように，Yを完全に修飾しなければならない。

14. 矛盾する文脈宣言は，避けなければならない。Pは，ポインターの名前としてよく用いられるので，省略時解釈によって別のデータ・タイプの性質が想定されないようにしなければならない。たとえば，

```
DCL B BASED (P),
    :
  P AUTO,
  .
  .
  .;
```

Pの明示宣言が最初にコンパイラーによって処理され，省略時解釈の属性，FLOAT と DECIMALがつけ加えられる。したがって，Pの文脈宣言は，矛盾したものになる。

15. 基底付変数をパラメーターとして使用することはできない。

PROCEDURE/ENTRYステートメントのオプション

OPTIONS オプションのオプションと，その他のオプション(RECURSIVE など) とを区別するよう，注意しなければならない。たとえば，次のステートメントは，

```
P:   PROC OPTIONS (MAIN,RECURSIVE);
```

正しくない。これは次のようにしなければならない。

```
P:   PROC OPTIONS (MAIN) RECURSIVE;
```

割当てと初期値化

1. 変数がアクセスされるとき，その値は，以前にそれに割り当てられたもので，その変数の属性と矛盾しないものであるとみなされる。この想定が正しくないときは，プログラムは誤ったデータを用いて先に進むか，あるいはプログラム割込みが起こる。このようなことは，変数の初期値化がうまくゆかなかったときや，次のいずれかのやりかたで変数がセットされたときに起こる可能性がある。

   a. UNSPEC 擬似変数を用いて

   b. レコード方式の入力を用いて

   c. 文字ストリングの上にピクチャーを定義づけしてからその文字ストリングに割り当て，さらにそのピクチャーをアクセスすることによって

   d. 別のプロシージャーで割り当てられた変数を，パラメーターの属性と一致させずに，引数として渡すことによって。

   e. 異なる属性をもっているが，同じ位置にあるような基底付変数に割り当てることによって。

変数の初期値化がうまく行なわれないと，実行時には，予期しないような値をもつ変数が作られる結果になる。この値がゼロであるとみなしてはならない。

添字の初期値化がうまく行なわれなかったときは，プログラムをデバッグするときにSUBSCRIPTRANGE を解禁すれば，発見することができる(初期値化されなかった値が添字の範囲内になければ)。

2. 初期値化されなかった変数または配列を出力しようとすると，当然のこととして，データ割込みが起こる。たとえば，

```
DCL A(10) FIXED;
A(1)=10;
PUT LIST (A);
```

データ割込みを避けるには，割当てステートメントの前で配列を初期値化しなければならない。したがって，次のようにする。

```
A=0;
```

この問題は，初期値化されなかった配列に対するCHECKシステム処置の結果として起こる可能性もある点に注意していただきたい。上の例の中で，この配列に対してCHECK条件が解禁されており，システム処置がとられたとすれば，その結果や，プログラムの終了の仕方は，予測できない。PUT DATAが使用されているときにも，同じ問題が起こる。

3. =(割当て)と=(比較)の違いに注意すること。次のステートメントは，

```
A=B=C;
```

「BとCを比較して，その結果('1'Bまたは'0'Bのどちらか)を，必要に応じてタイプ変換をした上でAに割り当てよ」という意味である。

4. 変換を含む割当ては，できる限り避けるべきである(この章の後の方の「算術演算と論理演算」の項の第2セクションを参照のこと)。

5. 固定長ストリングへの割当てまたは初期値化の場合：割り当てられる値がストリングよりも短いと，この値の右側にブランク(文字ストリングのとき)またはゼロ(ビット・ストリングのとき)が補なわれて拡張される。たとえば，

```
DCL A CHAR(6),
    B CHAR(3) INIT('CR');
    A=B;
```

上記のステートメントが実行されたあと，BにはCRb，AにはCRbbbbが入っている。

6. 割当てステートメントまたはその他の単一ステートメントの中で，構造の配列の断面を参照することはできない。構造の配列全体，または1つの要素は参照できるが，断面は参照できない。

7. SIZEが禁止されているときは，SIZEを引き起した割当ての結果は，予測できない。

FIXED BINARY：ここでの割当ての結果－－たとえば，原始言語による割当てや，パラメーターの突合せによって暗示される変換などを含む－－は，FIXEDOVERFLOWを起こす可能性がある。

FIXED DECIMAL：割込みを起さずに，最も近いバイトへの切捨てが起こる可能性がある。目標フィールドの精度が偶数のときは，最高位のバイトに余分の数が挿入される。

算術演算と論理演算

1. 式の評価の規則，とくに演算の優先順位に関して，充分注意する必要がある。次にあげる例は，起りがちな誤りの種類を示したものである。

X>Y|Zは，X>Y|X>Zと同じではないが，(X>Y)|Zとは同じである。

X>Y>Zは，X>Y&Y>Zと同じではないが，(X>Y)>Zとは同じである。

同じ優先順位をもつ一連の演算はすべて，左から右の順に評価されるが，**，接頭語＋，接頭語－，および¬は例外で，これらは右から左の順に評価される。したがって，次のステートメントは，

```
A=B**-C**D;
```

以下のものに等しい

```
A=B**(-(C**D));
```

括弧は通常，優先順位の規則を修正するために用いられる。しかし，安全手段として，あるいは演算を明確にするために余分な括弧を使用するのも便利である。

2. 変換は，わかりやすい規則にしたがって行なわれる。不要なトラブルを避けたいならば，この規則をよく理解しておく必要がある。変換の影響についてのいくつかの例を次にあげる。

a. DECIMAL FIXEDからBINARY FIXEDへの変換に，小数部が含まれていると，予期しないような結果になる可能性がある。

```
DCL I FIXED BIN(31,5) INIT(1);
    I = I+.1;
```

この場合，Iの値は1.0625になるが，これは，.1がFIXED BINARY(5,4)に変換されるので，その2進数の近似値は0.0001B(4捨5入なし)になるためである。これに相当する10進数は，.0625Bである。.1のかわりに .1000 を指定すれば，もう少しよい結果が得られる。

b. 文字ストリング・データに対して算術演算が行なわれるときは，最高の精度が(15,0)の固定10進数に中間結果が貯えられる。

```
DCL A CHAR(6) INIT('123.45');
DCL B FIXED(5,2);
B=A;   /*B HAS VALUE 123.45*/
B=A+A; /*B HAS VALUE 246.00*/
```

c. 10進定数からビット・ストリングへの割当ては，算術形式からビット・ストリングへの変換の規則によって，影響を受ける。

```
DCL A BIT(1),
    D BIT(5);
A=1; /*A HAS VALUE '0'B*/
D=1; /*D HAS VALUE '00010'B*/
D='1'B; /*D HAS VALUE
          '10000'B*/
IF A=1 THEN GO TO Y;
        ELSE GO TO X;
```

Aへの割当てが行なわれると，次のような一連の処置がとられるので，Xにブランチする。

(1) 10進定数の1は，FIXED DECIMAL(1,0)とみなされ，FIXED BINARY(4,0)という属性をもつ一時記憶域に，0001Bという値をとって割り当てられる。

(2) 次に，この値は長さ(4)のビット・ストリングとして扱われるので，'0001'Bになる。

(3) その結果のビット・ストリングがAに割り当てられる。Aは長さ1と宣言されているが，割り当てられる値は長さ4をもっているので，右端で切捨てが行なわれ，Aの最終値は'0'Bになる。

IFステートメントの中で比較演算をするために，'0'Bと1がFIXED BINARYに変換され，算術的に比較される。これらは等しくないので，A=1の関係は「偽」という結果になる。

Dに最初に割り当てるときには，Aについてのべたものとほぼ同じ一連の処置がとられる。ただし，Dについては，割り当てられる値の長さよりも1だけ多い長さが宣言されているので，右側にゼロが補なわれて値が拡張される。

d. 文字ストリングに算術値を割り当てるときは，F節「データ変換と式の値の計算」のところで述べる規則にしたがった変換が行なわれる。

例1

```
DCL A CHAR(4),
    B CHAR(7);
A='0';    /*A HAS VALUE '0bbb'*/
A=0;      /*A HAS VALUE 'bbb0'*/
B=1234567; /*B HAS VALUE
             'bbb1234'*/
```

注: 負符号の可能性，10進数または2進数の小数点，および小数点の前に先行ゼロを1桁置く，などの余裕をとるためには，3桁のブランクが必要である。

例2

```
DCL CTLNO CHAR(8) INIT('0');
DO I=1 TO 100;
   CTLNO=CTLNO+1;
   .
   .
   .
END;
```

この例では，次のような一連の処置がとられるため，変換エラーが起こる。

(1) CTLNOの初期値，すなわち'0bbbbbbb'がFIXED DECIMAL(15,0)に変換される。

(2) FIXED DECIMAL(1,0)とみなされる，10進定数の1が加えられる。加算の規則にしたがって，結果の精度は(16,0)である。

(3) 次にこの値は，CTLNOに割り当てし直すための準備として，長さ18の文字ストリングに変換される。

(4) CTLNOの長さは8なので，割当ての結果，右端で切り捨てられる。したがって，CTLNOの最終値は，全体がブランクになる。この値は，2回目のループの反復では算術タイプにうまく変換することができない。

e. FIXED の除算は，予期しないような桁あふれや切捨てを起すことがある。たとえば，次の式は，

```
25+1/3
```

5.33...3という値になる。25.33...3という結果を得るためには，次のように書く必要がある。

```
25+01/3
```

これは，書かれた通りの精度とスケール因数を定数がもっているのに対して，FIXED除算の結果は，そのシステムで決められた最大の精度をもつ値になる，ということである。2つの評価の結果は，次のようになる。

項目	精度／スケール因数	結果
1 3 1/3 25 25+1/3	(1,0) (1,0) (15,14) (2,0) (15,14)	1 3 0.33333333333333 25 5.33333333333333 （左で切り捨てられる；禁止されていない限りFIXEDOVERFLOW が起こる）
01 3 01/3 25 25+01/3	(2,0) (1,0) (15,13) (2,0) (15,13)	01 3 00.3333333333333 25 25.3333333333333

かわりに，次のようにしてPRECISION 組込み関数を使用してもよい。

```
25+PREC(1/3,15,13)
```

f. ピクチャー変数に割り当てられるときにだけ，ピクチャーの検査が行なわれる。

```
DCL A PIC'999999',
    B CHAR(6) DEF A,
    C CHAR(6);
B='ABCDEF';
C=A; /*WILL NOT RAISE CONV
        CONDITION*/
A=C; /*WILL RAISE CONV*/
```

次の例についても注意していただきたい（A，B，Cは，上記と同じに宣言されている）。

```
A=123456; /*A HAS VALUE
             123456*/
          /*B HAS VALUE
             '123456'*/
C=123456; /*C HAS VALUE
             'bbb123'*/
C=A; /*C HAS VALUE '123456'*/
```

g. 偶数の精度（P，Q）が宣言されているFIXED DECIMAL 要素は，最高位のバイトがゼロでないこともあり得るので，実際の精度が（P+1，Q）になることがある。このような影響をさけるために，SIZE 条件を使用することができる。

```
DCL (A,B,C) FIXED DECIMAL (6,0);
    ON SIZE;
      .
      .
      .
(SIZE): A = B + C;
```

このようにすると，割当ての後，Aの最高位のバイトは，確実にゼロになる。

DOグループ

1. DOステートメントに適用される条件接頭語の有効範囲は，そのステートメント自体の実行に限られており，グループ全体の実行には適用されない。

2. 初期値化のときに最終条件が満足されてしまったときは，反復DOグループは実行されない。

```
I=6;
DO J=I TO 4;
   X=X+J;
END;
```

このグループでは，BY 1が暗示されているので，Xは変えられない。逆方向に反復することもできるので，BY−1が指定されているときは，反復は3回行なわれる。

3. DOステートメントの中の式は，変数ではなく，式と同じ性質をもつ一時域に割り当てられる。たとえば，

```
DCL A DECIMAL FIXED(5,0);
      A=10;
      DO I=1 TO A/2;
      .
      .
      .
      END;
```

A／2は，10進数の精度（15，10）をもっており，これは2進数に変換されると（Iとの比較のために），2進数（31，34）になるので，このループは実行されない。

DOステートメントを次のように置き換えると，反復が5回行なわれる。

```
ITEMP=A/2;
DO I=1 TO ITEMP;
```

または

```
DO I=1 TO PREC(A/2,6,1);
```

4. DOグループは，オン・ユニットとして使用することはできない。ステートメントのグループに対しては，開始ブロックを使用しなければならない。

5. 反復DOグループの上限と下限は，関係する変数がグループの中で再割当てされるときでも，1回だけ計算される。これは，BY 式にも適用される。

   関係する変数に割り当てられる新しい値は，DOグループが再び開始されるときだけ，効力を発する。

6. 制御変数とWHILE オプションの両方をもつDOグループの中では，反復を行なうことができると判定された（制御変数の値から）あとでだけ，WHILE 式の評価とテストが行なわれる。たとえば，次のグループは，多くて1回だけ実行される。

```
DO I=1 WHILE(X>Y);
.
.
.
END;
```

7. Iは，DOグループの中で制御変数としてよく用いられる。たとえば，

```
DO I=1 TO 10;
```

   この暗示宣言の有効範囲内で，Iがポインターとして文脈的に宣言されることがある。たとえば，

```
DCL X BASED(I);
```

   この2つのステートメントは矛盾しているので，診断メッセージが出される。

   Iがポインター変数のときは，次のいずれかの方法を用いたときだけ，DOグループの中で使用することができる。

```
a. DCL (I, IA, IB, IC) POINTER;
      .
      .
      .
    DO I=IA,IB,IC;
b. DCL (I, IA) POINTER;
      .
      .
      .
    DO WHILE(I=IA);
```

データ集合体

1. 配列の計算は，反復計算を指定するための便法を考えるべきである。たとえば，

```
DCL A(10,20);
.
.
.
A=A/A(1,1);
```

   は，次のものと同じ効果をもっている。

```
DCL A(10,20);
.
.
.
DO I=1 TO 10;
DO J=1 TO 20;
A(I,J)=A(I,J)/A(1,1);
END;END;
```

   この場合，最初の反復ではA（1，1）に対する値が1になるので，その効果は，A（1，1）の値が変わるだけであることに注意していただきたい。A（1，1）のもとの値によってAの各要素を除算したい場合は，次のように書けばよい。

```
B=A (1,1);
A=A/B;
```

   または，かわりに次のようにしてもよい。

```
DCL A(10,20);
    B(10,20);
.
.
.
B=A/A(1,1);
```

2. 配列の乗算の効果に注意していただきたい。

```
DCL (A,B,C) (10,10);
.
.
.
A=B*C;
```

これは，行列式の乗算の効果とは違っており，次のようにしたのと同じである。

```
DCL (A,B,C) (10,10);
.
.
.
DO I=1 TO 10;
DO J=1 TO 10;
A(I,J)=B(I,J)*C(I,J);
END; END;
```

ストリング

1. SUBSTR 擬似変数を用いて，長さの変わるストリングに割当てをしても，ストリングの長さはセットされない。長さの変わるストリングは，初めに長さが定義されないので，そのストリングへの割当てがすべて，SUBSTR 擬似変数を用いて行なわれても，ストリングは依然として不定長であり，他の変数に割り当てたり，書き出したりしてもうまくゆかない。

2. ストリング式の中間結果の長さが，32767バイトを越えていないか確認しなければならない。長さの変わるストリングの場合は，目的時に長さのチェックが行なわれないので，とくにこのことがいえる。

関数と擬似変数

1. 擬似変数としてUNSPEC を使用するときは，右辺の式はビット・ストリングに変換される。したがって，この式は，このような変換に関して無効なものであってはならない。たとえば，この式が，0または1以外の文字を含む文字ストリングのときは，変換エラーが起こる。

ON条件とオン・ユニット

1. ONステートメントを正しい位置に置くよう注意しなければならない。あるステートメントによって指名の条件がひき起されたときに，指定の処置を適用したいときは，そのステートメントが実行される前にONステートメントを実行しなければならない。次のステートメントの場合，

```
GET FILE (ACCTS) LIST (A,B,C);
ON TRANSMIT (ACCTS) GO TO TRERR;
```

最初のGET操作中に転送エラーが起ったときは,ERROR条件がひき起されるが，所要のブランチはとられない（前にONステートメントが適用されていないとすれば）。さらにまた，GETステートメントが実行されるたびに，その後でONステートメントが実行される。

2. GO TOステートメントを用いて，オン・ユニットに入ることはできない。故意にオン・ユニットを実行するためには，SIGNAL ステートメントを使用することができる。

3. 変換が正しくないために入った（SIGNAL の反対）CONVERSION オン・ユニットでは，正しくない文字を変えるか（ONSOURCE またはONCHAR 擬似変数によって），またはGO TO ステートメントによって終了させなければならない。このようにしていないときは，システムがメッセージを印刷し，ERROR 条件をひき起す。

4. AREA オン・ユニットからの正常な出口では，標準のシステム処置として，割振りがもう一度試みられる。その結果，再びオン・ユニットに入り，無限のループができる。これを避けるためには，EMPTY 組込み関数を用いたり，ポインター変数を変えたりするなどの方法によって，オン・ユニットの中で割り振られる量を修正しなければならない。

5. デバッギングのあとは，デバッギングの目的で解禁された条件（普通は禁止されているもの）を，NO-条件接頭語を含める方法ではなく，関連する接頭語をとり除くという方法で，禁止すること。たとえば，SIZE 条件の場合は,NOSIZE接頭語を追加するのではなく，SIZE 接頭語をとり除いて禁止する。前者の方法では，チェックが行なわれないようにするために余分のコードを生成する必要があるのに対して，後者の方法では，コンパイラーは，その条件をチェックするためのコードを排除することができる。

入力／出力

1. UNDEFINEDFILE 条件は，言語の属性が矛盾するとき（DIRECTとPRINTなど）ばかりでなく，次のような場合にも起こる。

   a. レコード・サイズよりもブロックサイズが小さいとき。

b. LINESIZEが，許される最大値を越えているとき。

c. INDEXED，またはREGIONAL(3)データ・セットの作成のときに，KEYLENGTH がゼロかまたは指定されていないとき。

d. INDEXEDデータ・セットの場合に，KEYLENGTH+KEYLOCの結果の値がレコードの長さを越えるような値をもつ，KEYLOC オプションを指定しているとき。

e. STREAM データ・セットに対して，18バイトより短かいV形式論理レコードが指定されているとき。

f. FB-形式のブロック化レコードに対して，レコード・サイズの整数倍でないブロック・サイズが指定されているとき。

g. VB-形式レコードに対して，指定のブロック・サイズより少くとも4バイト少ないものでない長さの論理レコードが指定されているとき。

h. VSAMファイルを開くときに，正しくないパスワードが指定されたとき。

i. VSAMデータ・セットが開かれている間に，VSAMによってエラーが発見されたとき。

2. ファイルが，入力と出力の両方に使用されるときは，INPUTまたはOUTPUT 属性のどちらかをつけて宣言しなければならない。OPENステートメントで，必要なオプションを指定することができる。

3. 入力／出力のリストは，1対の括弧で囲まなければならない。反復リストも同様である。したがって，次のステートメントの中では，外側の括弧は2対必要である。

```
GET LIST ((A(I) DO I=1 TO N));
```

4. REGIONAL データ・セットをアクセスするための原始キーの最後の8バイトは，固定10進整数を表わす文字ストリングでなければならない。キーを作成するときは，算術形式から文字ストリングへの変換の規則について考慮しなければならない。たとえば，次のグループはエラーになる。

```
DCL KEYS CHAR(8);
DO I=1 TO 10;
   KEYS=I;
   WRITE FILE(F) FROM (R)
     KEYFROM (KEYS);
END;
```

Iの省略時解釈は，FIXED BINARY(15,0)であるが，これは，この算術値の文字ストリング表現を収容するために，8桁ではなく，9桁を必要とする。

5. ファイルを参照する入出力ステートメントの中でKEY，KEYFROM，または KEYTO オプションが用いられるときは，そのファイルはKEYED 属性をもっていなければならないことに注意していただきたい。

6. 標準ファイル名SYSIN および SYSPRINTは，GETおよびPUTステートメントの中でだけ暗示される。そのほかに，ONステートメントまたはレコード方式の入出力ステートメントなどで行なわれる参照は，明示的でなければならない。

7. PAGESIZE および LINESIZE は，ファイル属性ではない。つまり，これらは，ファイルの宣言に含めることはできない。これらは，OPENステートメントのオプションである。

8. 編集指示のデータ・リストがなくなると，次の形式項目が，それに対応するデータ項目を必要としていなくても，それ以上形式項目は処理されない。たとえば，

```
DCL A FIXED(5),
    B FIXED (5,2);
GET EDIT (A,B) (F(5),F(5,2),X(70));
```

X(70)形式項目は，処理されない。次のカードの最初の10桁のデータだけを読むために，SKIPオプションを使用することができる。

```
GET SKIP EDIT (A,B) (F(5),F(5,2));
```

9. データ・リストの中にある配列名または構造名によって表わされるデータ・リストの数は，配列または構造の要素の数と同じである。したがって，2つ以上の形式項目が形式リストの中にあるときは，連続する要素は，連続する形式項目と突き合わせられる。たとえば，

```
DCL 1A,
      2 B CHAR(5),
      2 C FIXED(5,2);
 :
PUT EDIT (A) (A(5),F(5,2));
```

Bは，A(5)項目と突き合わせられ，Cは，F(5,2)項目と突き合わせられる。

10. 配列は，行を主とした順序で転送される(たとえば，A(1,1)，A(1,2)，A(1,3)，…A(2,1)，… など)

11. GET DATAおよびGET LIST に対する入力データとして用いられるストリングは，引用符で囲まなければならない。

12. 48文字セットでのセミコロンの表現(,.)は，データ指示入力ストリームの中では，セミコロンとして認められない。11−8−6穿孔を使用しなければならない。

13. ユーザーは，PUT DATA;の制約(つまり，データ・リストをもたない)に注意しなければならない。ONステートメントといっしょに使用されたときの効果は，PUT DATAに知られているデータはオン・ユニットの地点で知られているデータであるため，制約を受ける。

次のONステートメントが，

```
ON ERROR PUT DATA;
```

外側のブロックで使用されている場合は，内側のブロックの変数は知られていないため，それらはダンプされないことに注意しなければならない。したがって，デバッギング中に，内側のすべてのブロックの中でオン・ユニットを繰り返すことはよい方法である。

PUT DATA ステートメントの実行中にエラーが起り，しかもこのステートメントがERROR オン・ユニットの中にある場合は，記憶域がなくなるまでダンプを行なう。これは，機械時間と印刷を無駄にすることになるので，活用化されたあとは，ERRORオン・ユニットをオフにすべきである。

```
ON ERROR PUT DATA;
```

上のステートメントのかわりに，次のコードを使った方がよい。

```
ON ERROR BEGIN;
         ON ERROR SYSTEM;
         PUT DATA;
         END;
```

データ・リストのないPUT DATA を使用したときは，プログラム内のその地点で知られているすべての変数が，指定のファイルにデータ指示出力形式で転送される。しかし，この機能を使用するとき，ユーザーは，次の点に注意しなければならない。

a) 初期値化されないFIXED DECIMAL データは，CONVERSION条件またはデータ割込みを起こす可能性がある。

b) 割り当てられていないCONTROLLED データは，任意の値を印刷するので，FIXED DECIMAL の場合は，CONVERSION条件またはデータ割込みを起こす可能性がある。

14. READ SET またはLOCATE SET の中でセットされたポインターは，そのファイルに対して次の操作が行なわれたあと，またはCLOSE ステートメントの後では，無効である。OUTPUT ファイルの場合は，WRITE と LOCATEステートメントは，自由に混ぜることができる。

緩衝域に読み込まれたレコードを書き直して(SEQUENTIAL UPDATE ファイルを指定するREAD SET ステートメントを用いて)，更新する必要がある場合は，FROMオプションをもたないREWRITE ステートメントを使用することができる。READ SETのあとでREWRITE を与えると，その結果つねに，最後の緩衝域の内容がデータ・セットに書き直されることになる。たとえば，

```
3  READ FILE (F) SET (P);
        :
5  P->R = S;
        :
7  REWRITE FILE (F);
        :
11 READ FILE (F) INTO (X);
        :
15 REWRITE FILE (F);
        :
19 REWRITE FILE (F) FROM (X);
```

注:

ステートメント7は，緩衝域の中で更新されたレコードを書き直す。

ステートメント15は，データ・セット上のレコードを全く変えない。

ステートメント19は，前にREADステートメントがないので，ERROR になる。

レコードの転送が試みられるまで，LOCATE ステートメントのKEY条件をチェックすることが不可能な場合がある。

これは，

REGIONAL (3) ファイル上にレコードを出力するための余地が，指定の区域内にないときである。この条件を起したレコードも，現地のレコードも転送されない。

ファイルが閉じられる前に実行された最後の入出力ステートメントがLOCATE の場合は，ファイルが閉じられるまでレコードは転送されない。

したがって，CLOSE ステートメントによって，条件がひき起されることがある。

15. 記憶域が割り振られていない基底付記憶域に対して，目的時に参照が行なわれると，予測できない割込み（記憶保護，アドレス指定，または指定）が起こる可能性がある。

16. エリア，ポインター，オフセットおよび構造で，それらのいずれかを含むものは，STREAM 入出力で使用することはできない。

17. 基底付記憶域が解放されたとき，関連するポインターには，後に立つ情報はもう入っていない。

18. エリアの中で割り振られた基底付変数は，そのエリアの中で解放しなければならない。たとえば，

```
DCL A AREA, B BASED (X);
ALLOCATE B IN (A);
     .
     .
     .
FREE B;             /* INVALID */
FREE B IN (A);      /* VALID   */
```

19. ASCII データ・セットから読み込まれたブロックの中にあるレコードの最初のバイトは，必ずしも緩衝域の中でダブルワード境界に位置合せされない。ブロック接頭語は，ダブルワード境界に合わせられるが，レコードの境界合せは，ブロック接頭語の長さによって異なる。

第17章:　言語間の連絡機能

PL／I最適化コンパイラーによってコンパイルされたプログラムと，次にあげるコンパイラーまたはプログラム・プロダクトのいずれかによってコンパイルされ，対応するライブラリーを用いて実行されるプログラムとの間の，実行時における連絡は，PL／I言語間機能を用いて行なうことができる。コンパイラー，プログラム・プロダクト，およびライブラリーはすべて，DOS用にIBMで開発されたものである。

COBOL(D) レベル D	360N-CB-452
COBOL ANS レベル F	360B-CB-482
	5736-EB2，5736-LM2
COBOL ANS SUBSET	5736-CB1
FORTRAN IV レベル F	360N-FO-479
FORTRAN IV ライブラリー・サブプログラム	360N-LM-480
DOS ライブラリー・オプション	5746-LM3

PL／I のプログラムと，FORTRANまたはCOBOL コンパイラーのいずれかによってコンパイルされたプログラムとの間の連絡は，次の2つの方法のいずれかによって行なうことができる。

1. PL／IおよびCOBOL／FORTRAN ルーチンに対して共通のデータ・セットを用いることによって。

2. PL／I ルーチンからCOBOL／FORTRAN ルーチンを呼び出し(またはその逆)引数，または静的記憶域の形式でデータを渡すことによって。

PL／I とCOBOL ルーチンの間の連絡に共通のデータ・セットを使用する場合は，ENVIRONMENT属性のCOBOL オプションが必要になることがある。さらに詳しい点については，第12章の「データ交換」の項をみていただきたい。

PL／I プロシージャーでは，CALL ステートメントを用いることによってCOBOL ルーチンを呼び出したり，CALL ステートメントまたは関数参照を用いることによってFORTRAN ルーチンを呼び出したりすることができる。かわりに，COBOL またはFORTRAN の主プログラムまたは主ルーチンの中で対応する言語機能を用いることによってPL／I プロシージャーを呼び出すことができる。呼出しのときに引数を渡すことができるし，また，関数参照に対して値を返すことができる。

FORTRAN における COMMON ブロックの記憶域は,PL／IにおけるSTATIC EXTERNAL 変数の場合の記憶域と同じである。COMMONブロックとSTATIC EXTERNAL 変数に同じ名前が与えられている場合は，PL／I で2つの同一のSTATIC EXTERNAL 変数に対するのと同じようにして，同じブロックの記憶域がこれらに割り振られる。一方の変数に値が割り当てられると，他方にも同じ値が割り当てられる。COBOL では，このような同一扱いは行なわれない――COBOL の変数は，引数またはパラメーター以外に，PL／I 変数との間に共通の記憶域を共有することはできない。

言語間の連絡機能は，PL／I のコンパイラーによって全面的に準備される。これらの機能は，呼出し側または呼び出される側のPL／I プロシージャーの中で適当な言語項目を指定することによって，利用することができる。一般に，既存のCOBOL または FORTRAN プログラムまたはルーチンは，言語間で使用するために，修正したり，コンパイルし直したりする必要はない。また，新しいプログラムまたはルーチンを書くときは，それらの言語で書いて，前の通りにコンパイルすればよく，言語間の連絡を予想して書く必要はない。したがって既存のCOBOLまたはFORTRAN の適用業務プログラムは，PL／I プロシージャーを用いることによって拡張することができるが，一方，COBOL または FORTRAN ライブラリーは，新しいPL／Iプロシージャーまたは既存のPL／Iプロシージャーで使用することができる。

注:　本章の説明の中で「ルーチン」というときは，COBOL サブプログラム，または FORTRAN サブルーチンまたは関数(FORTRAN ライブラリー関数も含めて)もその中に含まれる。これらの言語の中でサブルーチンや関数を取扱う際に存在する規則は，通常そのまま適用され，言語間で使用するために修正されることはない。とくに，FORTRAN の関数の場合に，1個以上の引数を渡さずに呼出しをすることはできないという制約は，呼出しがPL／I ルーチンから行なわれるときにもやはり適用される。

言語間の機能

言語間の機能を使用する際に，COBOLやFORTRAN についての詳しい知識をもっている必要はないが，PL／I と他の2つの言語におけるデータの構成の等しさについて注意する必要はある。引数／パラメーターの突合せをするために，これらの等しさについて理解しなければならない。

引数とパラメーターの突合せをするとき，言語間の機能は，等価のデータ構成に対するマップ作成の相違点を自動的に解決する。プログラマーは，必要に応じてこの処置を打ち消すこともできる。

呼び出されたCOBOL またはFORTRAN ルーチンをカバーするために，PL／I の割込み処理を拡張するための機能も用意されている。

COBOLまたはFORTRAN ルーチンへの引数の受渡し

引数がCOBOLまたはFORTRANのルーチンに渡されるときは，PL／Iの普通のやり方で，つまり，関連する入口宣言のパラメーター記述子リストまたは引数そのものから，データ・タイプが決定される。しかし，言語間機能は，その引数に対するアドレス体系が，呼び出された言語で使用されるものであるか，また，他の点で必要がなければ，渡される集合体のマップが，呼び出された言語で使用されているものであるかを確める。PL／Iに備わっている言語間機能は，呼び出されたルーチンの中のパラメーターをみることはできないので，プログラマーは自己の責任において，呼び出されたルーチンの中にあるパラメーターが，データ・タイプと構成の面で，PL／Iでの引数の記述に対応するようにしなければならないことに注意していただきたい。

構造または配列引数のマップが，PL／Iと，呼び出された言語の両方で同じであることを，コンパイル時にPL／Iコンパイラーが判断できる場合は，引数は，呼び出されたルーチンに直接渡される。しかし，このようなマップの形が同じでない場合は，言語間機能は，みかけ引数を渡すよう準備する。この場合の見かけ引数は，呼び出される言語の規則にしたがってマップされる。K節「データ・マップの作成」を参照のこと。

FORTRANまたはCOBOLに渡される引数のPL／Iのデータ・タイプが，これらの言語の中で該当するものをもたない場合は，コンパイル時に，警告メッセージが出される。実行されたときの結果はどうなるかわからず，異常停止になることもある。

データ・タイプ： PL／Iでは，COBOLやFORTRANよりも，データ・タイプの数が多い。したがって，これらの言語の中で該当するタイプをもたないものもある。PL／Iのデータ・タイプがCOBOLまたはFORTRANの中で対応するものをもつ範囲，したがって，引数として渡すことのできる範囲が，ここに要約してある。

問題データ： PL／Iのデータ・タイプの多くは，COBOLまたはFORTRANのいずれかに，該当するタイプをもっている。PL／I－COBOLおよびPL／I－FORTRANの間で対応するデータの表が，それぞれ「COBOL インターフェース」と「FORTRAN インターフェース」という題目で以下に示してある。

プログラム制御データ： 呼び出されたCOBOLまたはFORTRANルーチンに対して，プログラム制御データ・タイプの引数を渡すことができる。しかし，呼び出されたルーチンの中では，入口引数しか，渡して使うことはできないので，そのルーチンがFORTRANルーチンのときだけ可能である。それ以外のデータ・タイプの引数は，次にPL／Iプロシージャーに渡す以外に，呼び出されたルーチンの中で使用してはならない。

注： 特定の入出力操作で使用されるファイルに対して，ENVIRONMENT属性の中でCOBOLオプションを指定することができる。このオプションをもつファイルは，ファイル引数またはファイル・パラメーターとして使用することはできない。ENVIRONMENT属性のCOBOLオプションの使いかたについては，第12章「レコード方式転送」の「ENVIRONMENT属性」の項をみていただきたい。

データのマップ： COBOLまたはFORTRAN ルーチンに引数をうまく渡すことができるようにするために，渡される実際の引数のマップは，そのパラメーターに対してCOBOLまたはFORTRANが想定するマップと一致していなければならない。

要素引数の場合は，引数とパラメーターの境界合せに互換性があるということが，唯一の条件である。PL／Iでは，変数の境界合せは，ALIGNEDとUNALIGNED属性によって決まる。COBOLとFORTRANでの，それに対応する指定は，次の通りである。

PL／I	COBOL	FORTRAN
ALIGNED	SYNCHRONIZED	通常の境界合せ
UNALIGNED	SYNCHRONIZE されない	対応するものなし

PL／Iの引数の境界合せは，データ・タイプと同様，パラメーター記述子リストまたは引数そのものから，推論される。SYNCHRONIZED COBOLパラメーター，またはFORTRANパラメーターに対して渡すことのできるのは，ALIGNEDの要素だけである。SYNCHRONIZEされていないCOBOLのパラメーターに対しては，ALIGNEDまたはUNALIGNEDのどちらの要素でも渡すことができる。これらの境界合せが互換性をもっているかどうか確めるのは，プログラマーの責任である。

データの集合体の場合，問題はさらに複雑である。たとえば，PL／IまたはCOBOLの構造は，上で述べたように，境界合せについて厳格さをもっている可能性がある。さらに，その各々のメンバーは，それ自体境界合せの厳格さをもっていることも，すべてのメンバーが同じ境界合せの厳格さをもっていることもある。各メンバーに対して必要な境界合せをするために，該当の言語のマップ作成の算法によって，パディング・バイトが挿入される。PL／Iの構造では，必要なパディングの量を最小にするために，可能な限り，境界合せは調整される。それに対して，COBOLの構造では，このような調整は起らない。その結果，PL／Iのマップ作成の算法によってマップが作られた構造と，COBOLの算法でマップが作られた構造とでは，主記憶域でのレイアウトが同じにならないこともある。

同様に，配列のマップも，PL／IとFORTRANとでは違っている。PL／Iは，2次元以上の配列を列を主とした順序で記憶するのに対して，FORTRANは，行を主とした順序でそれらを記憶する。したがって，2次元以上の配列の場合，PL／Iで要素を参

照すると，FORTRAN で同じ要素を参照するのに用いられる添字の順序を逆にすることによってとり出される。

言語間機能は，COBOLまたはFORTRANルーチンに引数として渡されるPL/Iのデータ集合体に対して見かけ引数を作ることによって，これらの問題を解決する。PL/Iの構造がCOBOLルーチンに引数として渡されるときは，双方の言語における引数のマップについて考慮される。マップが同じであるとコンパイラーが判断できる場合は，引数は，COBOL ルーチンに直接渡される。

しかし，マップが同じであるとコンパイラーが判断できない場合は，COBOL のマップの算法にしたがってマップされた，見かけ引数が作られる。PL/I構造のメンバーの値が，見かけ引数の中の対応するメンバーに割り当てられ，そのあとで，COBOLルーチンに引数として見かけ引数が渡される。PL/I プロシージャーに戻るとき，見かけ引数の中の値(変更されている場合も，されていない場合もある)が，もとのPL/I引数の対応するメンバーに割り当てられる。

同様に，PL/Iの配列がFORTRAN ルーチンに引数として渡されるときは，双方の言語における配列のマップが考慮される。双方の配列が1次元で，しかも連続する記憶域に入っており，同じに境界合せされている場合は，呼び出されたFORTRAN ルーチンに引数が直接渡される。双方の配列が1次元ではあるが，上記の条件に適っていないか，または2次元以上の場合は，FORTRANの配列の処理にしたがってマップされた見かけ引数が作られる。(実際には，このことは添字を逆にすることを意味している。)
PL/I の配列要素の値は，見かけ引数の中の対応する要素に割り当てられる。そのあとで，見かけ引数が，引数として FORTRANルーチンに渡される。PL/I プロシージャーに戻るとき，見かけ引数の中にある値(変更された場合も，されなかった場合もある)が，PL/I 引数の該当する要素に割り当てられる。

プログラマーは，データの集合体のマップを作り直すための言語間機能の効果を禁止したり，制限したりするための特定のオプションを指定することができる。たとえば，呼出しの際に数個の引数が渡される場合，プログラマーは，ある1つの引数に対してはこの機能を禁止し，別の引数にはこの機能を適用したり，あるいは第3の引数に対してこの機能を制限する，というようにすることもできる。

呼 出 し

COBOL またはFORTRAN ルーチンの呼出しは，COBOL またはFORTRAN ルーチンの入口点に対応する値をもつ入口定数または変数を指定する関数参照またはCALLステートメント(FORTRAN ルーチンの場合のみ)によって，行なわれる。
この入口定数または変数は，PL/Iプログラム内の入口宣言の中で，OPTIONS 属性の該当するオプションを使用することによって，COBOL またはFORTRANを呼び出すものとして識別しなければならない。また，プログラマーは，データ集合体のマップの作り直しを抑止するためのオプションや，COBOL または FORTRAN ルーチンの中で起こる特定の割込みをPL/I が処理できるようにするためのオプションも，指定することができる。

これらのオプションは，次の通りである。

COBOL: これは，指定された入口点がCOBOLルーチンの中にあることを指定するものである。

FORTRAN: これは，指定された入口点がFORTRAN ルーチンの中にあることを指定するものである。

NOMAP: これは，見かけ引数を作らないことを指定するものである。集合体引数は，呼び出されたルーチンに直接渡される。

NOMAPIN: これは，見かけ引数が作られるときに，集合体引数の値で初期値化しないことを指定するものである。

NOMAPOUT: 見かけ引数が作られる場合に，あとで戻るとき，見かけ引数の値を集合体引数に割り当てないことを指定するものである。

ARGn: これは，NOMAP, NOMAPIN, および NOMAPOUT の各オプションが適用される引数を指定するために，用いられる。ARGnが指定されていないときは，すべての引数にオプションが適用される。

INTER: これは，COBOL または FORTRAN 割込み処理機能によって処理されない COBOL またはFORTRAN ルーチンの実行中に起った割込みを，PL/I 割込み処理機能で処理することを指定するものである(この章の後の方の「割込み処理」の項を参照)。

NOMAPIN およびNOMAPOUT オプションを使用すると，コンパイラーは，不要の初期値化コードを省くことができるので，プログラムの効率が重要な問題であるときは，初期値化が不要ならば，これらのオプションを使用すべきである。

入口名を宣言するときは，次の点に注意しなければならない。

1. COBOL またはFORTRAN のどちらかだけ(両方は不可)を宣言の中に記入することができる。NOMAP，NOMAPIN，NOMAPOUTの各オプションの中，1つまたは2つ以上を同じ宣言の中に記入してもよい。

2. COBOLのサブプログラムは，値を返さないので，COBOLオプションといっしょにRETURNS属性を使用することはできない。

3. 言語間機能オプションといっしょに入口変数またはパラメーターを宣言することができる。

4. GENERIC 属性指定の中で，言語間機能オプションといっしょに入口名を記入することができる。

5. COBOL またはFORTRAN ルーチンの入口定数名は，1から8桁までである。9桁以上指定すると，左端の8桁だけがとられる。

例：

```
1. DCL COBOL ENTRY(CHAR(5))
          OPTIONS(COBOL INTER),
         COBOLB ENTRY (1,2 FIXED,2 FLOAT)
          OPTIONS(COBOL NOMAPIN),
         COBOLBXX (OPTIONS(COBOL)
          EXTERNAL ENTRY(...);
2. DCL FORTA ENTRY(FIXED)
          OPTIONS(FORTRAN) RETURNS
          (FLOAT (5));
3. DCL A EXTERNAL ENTRY(...) VARIABLE
          OPTIONS (FORTRAN),
       B OPTIONS(FORTRAN);
            :
       A=B;
       CALL A (...);

4. DCL A GENERIC COBOLZ
          WHEN(1 UNALIGNED, 2 FIXED,
          2 FLOAT),
            FORTZ WHEN(FIXED BINARY)),

         COBOLZ OPTIONS(COBOL),

         FORTZ OPTIONS(FORTRAN);

5. DCL A ENTRY;
   CALL X(A);
            :
   X:PROC(B);
   DCL B OPTIONS(COBOL);
6. DCL COBSUB ENTRY(…,…,…,)
         OPTIONS(COBOL,NOMAP(ARG1,ARG3);
            :
   CALL COBSUB(A, B, C);
            :
   CALL COBSUB (X, Y, Z);
```

PL／I プロシージャーへの引数の受渡し

COBOL またはFORTRAN からPL／Iプロシージャーに引数が渡されるとき，データ・タイプは，PL／I の普通の方法で，つまりパラメーターの宣言から，決められる。言語間機能は，そのパラメーターに対して用いられているアドレス体系がPL／I で使用されるものであること，およびそれ以外に必要がなければ，渡される集合体パラメーターのマップもPL／I で使用されるものであることを確かめる。PL／Iに備わっている言語間機能は，PL／Iを呼び出しているルーチンの中の引数をみることはできないので，プログラマーの責任において，PL／I に渡される引数が，データ・タイプの点でも構成の点でも，PL／I で宣言されているパラメーターと対応するよう，確認しなければならない。

データ・マップの作成： この状況は，PL／I によってCOBOLまたはFORTRAN が呼び出されたときに起こる状況とほぼ同じである。PL／I プロシージャーに入るときの引数のマップは，パラメーターをアドレスするときにPL／I が使用するマップに対応していなければならない。

要素引数およびパラメーターの場合，これは，SYNCHRONIZED またはSYNCHRONIZEDされないCOBOLの引数を，UNALIGNEDのPL／I パラメーターに渡してもよいこと，またはSYNCHRONIZED のCOBOL引数またはFORTRAN の引数を，ALIGNED PL／I パラメーターに渡すことができることを意味している。

COBOL またはFORTRAN の引数のマップが，PL／I のパラメーターのマップと違っているときの集合体引数とパラメーターの場合，言語間機能は，PL／I プロシージャーに渡される見かけ引数を作成することによって，問題を解決する。

見かけ引数は，PL／I の規則にしたがってマップされ，PL／I プロシージャーが呼び出される前にCOBOL またはFORTRAN引数のメンバーの値が，見かけ引数の対応するメンバーに割り当てられる。PL／I プロシージャーから戻るとき，見かけ引数のメンバーの値がもとの引数に割り当てられて戻される。

COBOLまたはFORTRANとPL/Iにおけるマップが同じであることを，コンパイラーが認識できる場合は，このような見かけ引数は作られない。かわりに，プログラマーは，見かけ引数が作られること，あるいはもとの引数と作られた見かけ引数との間で割当てが行われることを，オプションを使って禁止することができる。

呼出し

COBOLまたはFORTRANから呼び出されるPL/Iプロシージャーの中にある入口点は，対応するPROCEDUREまたはENTRYステートメントの中で該当するオプションを使って識別しなければならない，また，プログラマーは，データ集合体のマップの作り直しを抑止するオプションを指定することもできる。

COBOL: これは，その入口点を呼び出すことができるのは，COBOLルーチンだけであることを指定するものである。

FORTRAN: これは，その入口点を呼び出すことができるのは，FORTRANルーチンだけであることを指定するものである。

NOMAP: これは，見かけ引数を作らないように指定するものである。COBOLまたはFORTRANの集合体引数は，PL/Iに直接渡される。

NOMAPIN: これは，見かけ引数が作られるときに，集合体引数の値で初期値化しないことを指定するものである。

NOMAPOUT: これは，見かけ引数が作成されるとき，戻りの際にその値を集合体引数に割り当てし直さないことを指定するものである。

パラメーター・リスト: NOMAP, NOMAPIN, またはNOMAPOUTを適用するパラメーター（2つ以上も可）を，パラメーター・リストの中に指定することができる。

NOMAPINおよびNOMAPOUTオプションを使用すると，コンパイラーは，不必要な初期値化のコードを省くことができるので，プログラムの効率が重要な問題であるときは，これらのオプションを使用しなければならない。

PROCEDUREまたはENTRYステートメントをコーディングするときは，次のような点に注意しなければならない。

1. MAIN, COBOL, またはFORTRANのオプションは，1つのステートメントの中では1つしか指定することはできない。NOMAP, NOMAPIN, またはNOMAPOUTのオプションは，1つのステートメントの中で1つでも，2つ以上でも指定することができる。

2. プロシージャーに対するパラメーターに，ストリング，エリア，または配列が含まれている場合は，これらに対する長さ，サイズ，または限界を，10進整定数として指定しなければならない。

3. COBOLルーチンによって呼び出された入口点に対して，RETURNSオプションを指定することはできない。

4. NOMAP, NOMAPIN, またはNOMAPOUTオプションに対するパラメーター（2つ以上も可）は，引数リストの中で指定することができる。

例:

1.
```
P1:PROC(A,B,C) OPTIONS(FORTRAN
               NOMAPIN(C)
               NOMAPOUT(A));
     DCL A (3,4) FLOAT BIN(20),
         B FIXED BIN(31),
         C (5,6) FLOAT DEC(6);
```

2.
```
P2:PROC(R,S,T) OPTIONS(FORTRAN
                        NOMAP);
```

3.
```
P3:PROC(X,Y) OPTIONS(COBOL NOMAPIN
                    (X) NOMAPOUT(Y));
     DCL 1 X,2...,
         1 Y,2...;
```

共通域の使用

PL/Iプログラムの中の変数を，FORTRANルーチンの中にある変数と同じ記憶域ブロックに割り振ることができる。このようにすると，2つのルーチンの間で連絡するために，この変数を使用することができる。共通域の割振りは，PL/Iの変数をSTATIC EXTERNALと宣言し，FORTRANルーチンの中に同じ名前をもつCOMMONブロックを設定することによって，行なわれる。このようにすると，PL/Iの変数と，COMMONブロック内の変数の関係は，2つのPL/I外部プロシージャーの中に同じ名前をもつ2つのSTATIC EXTERNAL変数がある場合と等しくなる。共通域を使用する変数の数は，2つに制限されているわけではなく，別のいくつかのPL/Iプロシージャーの中にある同一のSTATIC EXTERNAL変数と，別のいくつかのFORTRANルーチンの中にある同一のCOMMONブロックとは，これらのプロシージャーおよびルーチンがすべて，連係編集されて1つのプログラムになる限りは，双方ともいくつでもいっしょに使用することができる。連係編集の方法については，DOS PL/I最適化コンパイラーに対するプログラマーの手引に記載されている。

STATIC EXTERNAL変数は，これらの属性に関連する通常のPL/Iの規則を守らなければならず，またさらに，COMMONブロッ

クの中の変数のデータ・タイプに相当するデータ・タイプのものでなければならない(対応するデータ・タイプの表については，この章の後の方にある「FORTRANインターフェース」を参照のこと)。また，PL/Iの変数は，対応するFORTRANのデータ・タイプの必要条件に合わせて，境界合せしなければならない。

PL/Iの変数は，INITIAL属性を用いて初期値化することができるが，FORTRANの変数は，BLOCK DATA サブプログラムを用いて初期値化することができる。一方のPL/I変数と他方のFORTRAN変数が同じ値に初期値化されていないときは，連係編集プログラムが最初に出あうプロシージャーまたはルーチンによって，すべての変数の初期値が決まる。PL/Iの変数と，対応するFORTRANの変数とを別の値に初期値化したり，一方だけを初期値化して他方を初期値化しない，ということは，別に誤りではない。

DEFINED属性を用いて，PL/I変数の上にさらに別の変数をオーバーレーすることは，定義づけされる変数がFORTRAN変数のデータ・タイプおよび境界合せの必要条件に合致している限り，許される。

PL/Iの変数とCOBOLの変数との間で共通域を使用することはできない。PL/IプロシージャーとCOBOLルーチンの間の連絡についてPL/Iに備わっている機能は，引数を渡すことだけである。

言語間の環境

PL/Iのプロシージャーを実行するためには，最初にPL/Iの環境を設定しなければならない。プログラムにPL/Iの主プロシージャーが含まれているときは，そのプログラムに最初に入るときにこの環境が設定される。主ルーチンがCOBOLまたはFORTRANの場合は，言語間機能によって，必要に応じて所要のPL/I環境が設定される。このセクションでは，諸言語間の文脈の中での規約や制約について説明する。

PL/I 環境の設定

プログラムの主ルーチンがPL/I主プロシージャーの場合は，プログラムに入るとき，PL/Iの環境が設定される。このプログラムにPL/IとCOBOLまたはFORTRANルーチンが混じって入っているときでも，PL/Iの記憶域の解放や，PL/Iファイルの閉止に関する規則が適用される。

プログラムの主ルーチンがPL/Iの主プロシージャーでないときは，PL/Iプロシージャーが最初に呼び出されるときに，PL/I環境が設定される。この環境の範囲には，PL/Iプロシージャーを呼び出したルーチン(図17.1参照)も含まれており，このルーチンが終了するまでこの環境は存在しつづける。この環境は，必要なだけ何度でも，再設定したり，終了させたりすることができる。PL/Iの環境が破壊されたときはつねに，PL/Iの被制御および基底付記憶域のすべてが解放され，PL/Iファイルはすべて閉じられる。

効率とプログラミングの便宜などの理由から，プログラムの実行中，PL/Iの環境を破壊する回数は，できる限り少くしなければならない。主ルーチンがPL/I主プロシージャーである場合や，PL/Iプロシージャー(その中に何を含んでいても)が主ルーチンから呼び出されている場合は，上記のことを確かめることができる。しかし，後者の場合，主ルーチンがFORTRANで書かれているときは，オペレーティング・システムに制御を戻すために最後のFORTRAN RETURNが実行されるときに，PL/I環境が正常に終了しないという欠点がある(この章の後の方の「FORTRANとCOBOLルーチンの終了」を参照のこと)。

FORTRANルーチンによる入出力の制約

SYSLSTを使用しているときは別として，PL/Iプロシージャーによって呼び出されたFORTRANルーチンが生成した入出力制御情報は，PL/I環境の情報の上に書き出される。

その逆に，入出力ルーチンを実行し，PL/Iプロシージャーを呼び出すFORTRANルーチンの入出力制御情報は，PL/I環境が設定されると，それによって上書きされる。しかし，このような問題を解決するために，各導入先で定義されるサブルーチンPLISAが用意されている。

PLISAサブルーチンは，PL/I環境を収容する記憶域を確保する。PLISAサブルーチンには，2つの引数が与えられる。最初の引数は，PL/Iの環境をとっておく変数の名前である。2番目の引数は，記憶域のバイト数を指定する整数の定数である。次にあげるステートメントの例は，FORTRANルーチンの中で与えることもできる。PLISAサブルーチンは，PL/Iプロシージャーが呼び出される前に呼び出さなければならない。たとえば，

```
REAL*8 PLIENV (250)
   ・
   ・
CALL PLISA (PLIENV,2000)
   ・
CALL PLIPROC(X)
```

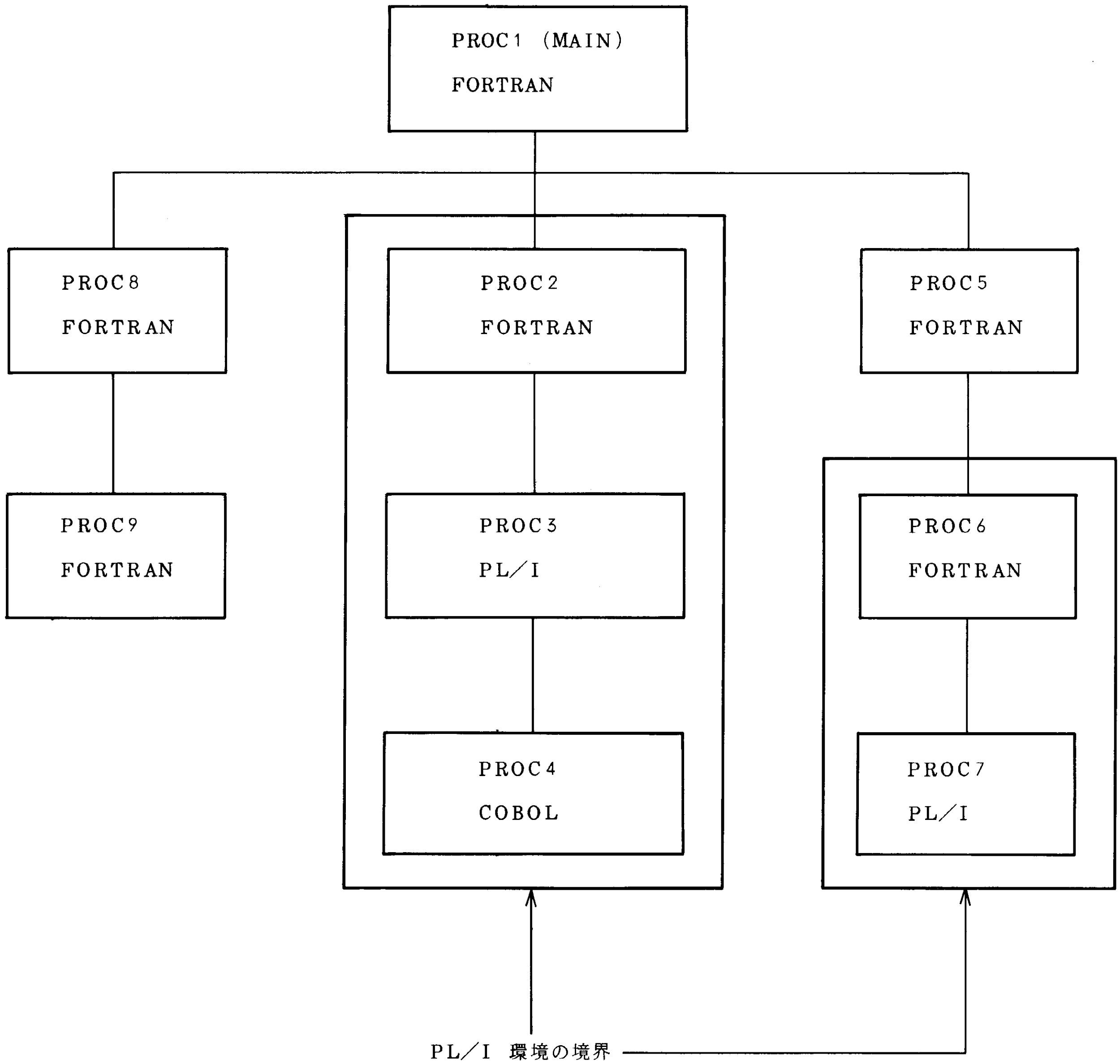


図17.1 PL/I 環境の範囲

注:

1. FORTRAN のデータ・タイプREAL *8は，この例では適切ではない。配列の次元が，同じバイト数を維持できるよう修正されていれば，どのようなデータ・タイプを指定してもよい。

2. この例で与えられているバイト数は，PL/I の環境で必要とされる最少限の記憶域の量とみなさなければならない。PL/I プロシージャーの実行中に必要な，環境情報の最大の量を決めるための簡便な方法というものはない。この処置を最も安全にする方法としては，FORTRAN の配列変数をできる限り大きくしておくことである。

FORTRAN ルーチンから呼び出されたPL/I プロシージャーがSYSPRINT ファイルを使用している場合は，FORTRAN ルーチンに戻る前にそのファイルが閉じられていない限り，最後の項目は転送されない。かわりに，最後の項目を転送するために，PUT SKIPステートメントを実行することができる。

同様に，PL/Iプロシージャーが FORTRANルーチンを呼び出しているときは，FORTRANルーチンから戻ってくるまで，最後の項

目は転送されない，したがって，FORTRANルーチンがSYSLSTを使用している場合は，SYSPRINT出力の最後の項目は，SYSLST出力の後に印刷される。このようなことを避けるには，FORTRANルーチンが呼び出される前に，SYSPRINTファイルを閉じるか，またはPUT SKIPステートメントを実行しなければならない。

割込み処理

COBOLおよびFORTRANルーチンは，実行中に起こる可能性のある特定のハードウエア割込みを処理することができるが，処理できない割込みもいくつかある。PL/Iの言語間連絡機能を使用すれば，COBOLまたはFORTRANルーチンで処理できないような割込みを，そのルーチンが動的に派生してきたPL/Iプロシージャーによって，処理することができるようになる。

プログラマーは，COBOLまたはFORTRAN入口名を宣言するとき，OPTIONS属性のINTERオプションを指定する。（この章の前の方の「COBOLまたはFORTRANルーチンへの引数の受渡し」のところで，INTERオプションの項も参照のこと。）このようにすると，呼び出されたCOBOLまたはFORTRANルーチンで処理されない割込みを，PL/Iのオン・ユニットかまたはPL/Iの標準システム処置のいずれかによって処理することができるようになる。PL/Iではオン・ユニットが設定されていれば，そのオン・ユニットは，それが作られたプロシージャーばかりでなく，そこから動的に派生したすべてのプロシージャーにも適用される。COBOLまたはFORTRANルーチンの実行中に，そのルーチンでは処理されないような割込みが起こり，しかもCOBOLまたはFORTRAN入口名に対してINTERオプションが指定されているPL/Iプロシージャーによってそのルーチンが呼び出された場合は，すべての呼出しプロシージャーにわたって，適当なオン・ユニットが探索される。何もみつからないときは，標準のシステム処置がとられる。INTERが指定されていないときは，探索は行なわれず，オペレーティング・システムの制御プログラムによって割込みが処理される。

PL/I環境の限界のところまで，呼出しチェインの中にあるすべてのルーチンが探索されることに注意していただきたい。したがって，この探索にCOBOLまたはFORTRANルーチンが含まれることもあり得ることである。PL/Iオン・ユニットだけが探索の対象となるので，この探索の結果にこれらのルーチンが何らかの影響を及ぼすことはない。

GO TO ステートメント

2つ以上の言語間の境界を越えて，つまり，あるルーチンが別の言語のルーチンを呼び出している呼出しとして言語間の境界が定義されているところを越えて制御を移すときに，GO TOステートメントを使用してはならない。オン・ユニットの中でプログラマーがGO TOステートメントを使用すると，このような制御の移動が不用意に開始されてしまうこともある。（オン・ユニットに入るということは，そのオン・ユニットに入る原因となったステートメントが実行されたブロックまたはサブルーチンの外へ制御を移すこととはみなされないことに注意していただきたい。オン・ユニットは，そこに入ってきたプロシージャーまたはルーチンに付加されたものとみなすことができる。このことは，COBOLまたはFORTRANルーチンからオン・ユニットに入った場合にも適用される）。次の例について考えてみよう。

```
P:PROCEDURE;
   DECLARE LAB LABEL(L1,L2)
   EXTERNAL,
   FORT ENTRY OPTIONS(FORTRAN
   INTER);
   ON ERROR GO TO LAB;
       .
       .
   CALL FORT;
       .
       .
   L1: .........;
       .
       .
END P;
       .
       .
Q:PROCEDURE OPTIONS(FORTRAN);
   DECLARE LAB LABEL(L1,L2)
               EXTERNAL;
       .
       .
   L2: .........;
       .
       .
   END Q;
```

CALL FORT; ステートメントが実行され，次にFORTがQをCALLするとしよう。さらにまた，Pの中に設定されたオン・ユニットへの入口を開始するQの中でエラーが起ったと仮定しよう。この段階では，オン・ユニットは，そこへ入ってきたプロシージャーに付加されるものとみなされるので，制御はまだプロシージャーQに残っている。LABの値がL1の場合は，GO TOブランチは，プロシージャーPに制御を戻し，そのときにQとFORTの間，およびFORTとPの間の言語間境界を越えることになるので，エラーになる。LABの値がL2のときは，GO TOが実行されても制御はプロシージャーQに残るので，エラーにはならない。FORTでの割込みによって，Qに入る前にオン・ユニットに入ったとすれば，LABの値がL1のときは，エラーにはならなかったはずである。この場合にただ1つだけ越えるであろう言語間境界は，FORTとPの間のFORTRAN-PL/I境界である。（この場合，プロシージャーQは活用状態ではないので，LABの値はL2であってはならない）。

FORTRANおよびCOBOLルーチンの終了

プログラム全体を終了させるステートメントを実行するか，または呼出しルーチンに制御を戻すことによって，ルーチンを終了させることができる。

プログラム全体を終了させるステートメントは，FORTRANではSTOP，COBOLではSTOP RUNである。これらは，PL/IのSTOPステートメントに相当している。これらのステートメントの効果は，いくつかの言語がまじり合ったプログラムの中でも変らず，プログラム全体を終了させる。

主ルーチンではないFORTRANまたはCOBOLルーチンが，PL/I環境の中でSTOP（FORTRAN）またはSTOP RUN（COBOL）を出した場合は，環境は終了しない。このことは，PL/I プロシージャーの中で開かれているファイルは，閉じられないこと，すなわち，出力データが失なわれる可能性があることを意味している。図17．1にある例について考えてみよう。PROC2の中のSTOPまたはPROC4の中のSTOP RUN は，PROC3 で開かれるファイルを閉じないし，また，PROC6 の中のSTOPは，PROC7の中のファイルを閉じない。

PL/I環境の内側にあるFORTRANサブルーチンまたは関数の中で実行され，しかもその環境の外側にあるルーチンに制御を戻すようなRETURN（いいかえれば，直接PL/Iルーチンを呼び出しているが，いずれかのPL/I ルーチンから動的に派生したのではないようなFORTRANルーチンの中にあるRETURNステートメント）が実行されると，PL/I 環境は終了し，動的に派生したPL/I プロシージャーの中にあるすべてのファイルは，閉じられる。しかし，FORTRAN主ルーチンの中にあるRETURNステートメントは，実質的にSTOP ステートメントの働きをする。つまり，ファイルを何も閉じずにオペレーティング・システムに制御を渡す。

PL/I環境の内部にあるCOBOL 主ルーチンがオペレーティング・システムに制御を戻すときは，この環境は正常に終了する。

COBOL インターフェース

PL/I-COBOL インターフェースにわたって引数/パラメーターの突合せをするときは，双方の言語におけるデータ・タイプとデータ構成の対応についての知識が必要である。COBOL のデータ・タイプとPL/I の対応づけが，図17．2に示されている。これらは，COBOL の引数またはパラメーターにそれぞれ結びつけられるPL/I のパラメーター記述子の中に記入すべきPL/I のデータ・タイプである。

PL/Iの要素変数を使ってCOBOLの項目を指定するためには，相等しいデータ・タイプについての知識で充分であるのに対して，相等しいデータ集合体（COBOLではグループ項目，PL/Iでは構造または配列）を指定するには，双方の言語のデータ構成の記述に関する知識が必要である。図17．3にあげた例は，COBOL のデータ集合体をPL/I のことばでどのように記述するかを示したものである。

COBOLでは，1つのグループ項目記述の中で，OCCURSクローズを3回以上繰り返すことはできない。このことは，COBOLルーチンに引数として渡す，PL/I の構造内における配列に制約を加えることになる。また，OCCURS クローズは，レベル01の記入項目で指定することができない。したがって，COBOL ルーチンに（または，COBOLルーチンから）渡されるPL/Iの構造では，レベル1の配列を使用することはできない。

エリアまたはビット・ストリング変数を含んでいるPL/I の構造を，COBOLルーチンへの引数として渡してはならない。このようにすると，診断メッセージが出され，構造のマップの作り直しは自動的に行なわれなくなる。

FORTRAN インターフェース

PL/I-FORTRAN インターフェースにわたって引数/パラメーターの突合せをするとき，およびPL/IとFORTRANの変数に対して共通域を使用するときは，双方の言語における，データ・タイプとデータ編成の対応についての知識が必要である。
FORTRANのデータ・タイプに相応するPL/Iのデータ・タイプが図17．4に示されている。これらは，FORTRAN の引数またはパラメーターにそれぞれ結びつけられるPL/I のパラメーターまたはパラメーター記述子の中，あるいはFORTRAN COMMONブロックと同じ名前をもつSTATIC EXTERNAL 変数の宣言の中に記入すべきPL/Iのデータ・タイプである。

COBOL				PL/I			
データ・タイプ	長さ（バイト）	境界合せ SYNCHRONIZED（ALIGNED）	境界合せ SYNCHRONIZEなし（UNALIGNED）	データ・タイプ	長さ（バイト）	境界合せ ALIGNED	境界合せ UNALIGNED
COMPUTATIONAL[1] 10進数 長さ： 1-4	2	ハーフワード	バイト	FIXED BINARY (15,0)（ハーフワード整数）	2	ハーフワード	バイト
5-9	4	フルワード	バイト	FIXED BINARY (31,0)（フルワード整数）	4	フルワード	バイト
10-18	8	フルワード	バイト	等しいものはない	—	—	—
COMPUTATIONAL-1	4	フルワード	バイト	FLOAT DEC(6)（短精度 浮動小数）	4	フルワード	バイト
COMPUTATIONAL-2	8	ダブルワード	バイト	FLOAT DEC(16)（長精度 浮動小数）	8	ダブルワード	バイト
COMPUTATIONAL-3	1[2]	バイト	バイト	FIXED DEC	1[2]	バイト	バイト
DISPLAY	いくらでも	バイト	バイト	CHARACTER[3]	いくらでも	バイト	バイト

[1] 10進数の長さは，ピクチャーの中の9の数に等しい。
[2] 最小の10進数値（つまり，1桁の数字）には，1バイトの長さが適用される。その他の値については，CEIL（（数字の桁数＋1）／2）バイトが与えられる。
[3] VARYING 属性を使用することはできない。

図17．2　COBOL-PL/I のデータの対応

```
01  A SYNCHRONIZED.                               1  A ALIGNED,
    02  B OCCURS 3 TIMES.                            2  B(3),
        03  C OCCURS 4 TIMES.                           3  C(4),
            04  D OCCURS 5 TIMES USAGE COMP-3              4  D(5) FIXED DECIMAL
                PIC S9999V999.                                          (7,3),
    02  E USAGE DISPLAY                              2  E,
        03  F PIC X(8).                                 3  F CHAR(8),
        03  G PIC 9(8).                                 3  G PIC '(8)9'
    02  DUMMY OCCURS 6 TIMES.                        2  H(6,7) FIXED BINARY(15,0);
        03  H OCCURS 7 TIMES USAGE COMP
                PIC S9999.
```

図17．3　COBOL および PL/I におけるデータ集合体の宣言

PL/I と COBOL の場合にくらべると，FORTRAN の場合，データ集合体は配列しかないので，PL/I と FORTRAN における同等のデータ集合体の指定は，簡単である。不連続の単次元の配列または多次元の配列を PL/I の引数として用いるときは，問題が起こる。

FORTRAN			PL/I			
データ・タイプ	長さ (バイト)	境界合せ[1]	データ・タイプ	長さ (バイト)	境界合せ	
					ALIGNED	UNALIGNED
INTEGER*2	2	ハーフワード	REAL FIXED BINARY(15)	2	ハーフワード	バイト
INTEGER*4	4	フルワード	REAL FIXED BINARY(31)	4	フルワード	バイト
REAL*4	4	フルワード	REAL FLOAT DEC(6) (実数 短精度 浮動小数)	4	フルワード	バイト
REAL*8	8	ダブルワード	REAL FLOAT DEC(16) (実数 長精度 浮動小数)	8	ダブルワード	バイト
COMPLEX*8	8	フルワード	COMPLEX FLOAT DEC(6) (複素数 短精度 浮動小数)	8	フルワード	バイト
COMPLEX*16	16	ダブルワード	COMPLEX FLOAT DEC(16) (複素数 長精度 浮動小数)	16	ダブルワード	バイト
LOGICAL*1	1	バイト	BIT(8)	1	バイト	ビット[2]
LOGICAL*4	4	バイト	BIT(32)	4	バイト	バイト

[1] 一般に，FORTRAN のデータは，このような境界合せをして主記憶域に入れられる。しかし，COMMON データは，つねにバイトの境界に合わせられる。COMMON 域の項目が，厳密性が段々少なくなってゆく順に貯えられていないときは，上記のことから指定割込みが起こることがある。

[2] UNALIGNED のビット・ストリングに必要な境界合せは，バイトでなくビットであるということが，PL/I－FORTRAN のデータ交換に影響することはない。これは，FORTRAN のストリングがつねに，バイトの整数倍になるためである。

図17．4　FORTRAN－PL/I のデータの対応

一般に，PL/I とFORTRAN の間で引数を渡すとき，言語間機能は，単次元の配列を呼出しプロシージャーに直接渡し，見かけ引数を作らない。しかし，不連続の記憶域にあるPL/I の単次元の配列が引数としてFORTRAN ルーチンに渡されるときは，言語間機能は見かけ引数を作り，この中に不連続の配列をマップする。そのあとで，見かけ引数が引数として渡される。戻るときには，見かけ引数の中の値が，配列の中の対応する要素に割り当てられる。

PL/I とFORTRAN ルーチンの間で渡される多次元の配列の場合は，NOMAPオプションが指定されていない限り，つねに見かけ引数が作られる。

PL/Iのビット・ストリング配列を，FORTRAN ルーチンに引数として渡す場合は，ストリングの長さとして8または32しか指定してはならない。それ以外の値を指定すると，警告メッセージが出され，配列のマップの作り直しは自動的に行なわれなくなる。同様に，FORTRAN変数と共通の記憶域をもつPL/I 変数の場合も，ストリングの値として指定できるのは，8か32だけである。

第Ⅱ部： 規則と構文法

A節： PL／I の構文表記法

本書においては，PL／I のステートメントまたは要素の他の組合せについて説明する場合，つねに，所定の表記法にもとづいてそのステートメントないし要素の組合せの書き方を図示してある。

この表記法は，どのプログラミング言語の構文法ないし構造を図示する場合にも使用することのできる標準化された表記法であり，この表記法がPL／I の一部を成しているわけではない。この表記法のもとでは，ある言語で使用できる一般的な表現様式が，簡単ではあるが正確に示される。ただし，この表記法は単に言語の諸要素の構造を説明するためのものであり，これら諸要素の意味を説明するためのものではない。具体的には，言語の諸要素の記入順序，所要の句読記号，指定可能なオプションがこの表記法のもとで示される。

以下の諸規則はこの表記法の利用方法を説明するためのものである。これらの規則はすべてのプログラミング言語について共通である。ただし，ここで取り上げてある例は，PL／I の例だけである。

1. 表記法変数は，プログラミング言語を構成している諸要素のある一般的な種別を表わすための名前である。表記法変数は，
   a． 邦字，英字の小文字，10進数字，およびハイフンで構成され，しかも文字で始まっていなければならない。

   b． 邦字または英字の小文字と英字の大文字との組合せによって構成されていなければならない。この場合，邦字または英字の小文字だけから成る部分と英字の大文字だけから成る部分とが1つずつあり，しかもこの2つの部分がハイフンでつながれていなければならない。

   本書においては，このような変数はすべて，上述の表記法にもとづいてこれを定義してあるか，あるいは意味の上からこれらの変数が表記法変数であるとわかるような形で定義されている。たとえば，

   (1) 数字。これは，0から9までの間の，ある1つの数字をこの場所に記入することを示している。
   (2) ファイル式。これは，あるファイルへの参照をこの場所に記入することを示している。

   (3) DOステートメント。これは，この場所にDOステートメントを記入することを示している。英字の大文字は言語のキーワードを表わしている。

2. 表記法定数は，図示されている文字を図示されている通りに記入することを示すためのものである。表記法定数は英字の大文字だけ，あるいはある1つの特殊文字によって構成される。たとえば，

   DECLARE　識別名　FIXED；

   これは，図示されている通りにDECLAREと記入し，その後に表記法変数「識別名」を記入し，次いで図示されている通りにFIXEDと記入し，さらにその後に図示されている通りの特殊文字（セミコロン，；）を記入することを示している（表記法変数「識別名」はどこか別の場所で定義される）。

3. 以下の規則の中には「構文上の単位」と言う用語が出て来る。この用語は次のいずれか一方を表わしている。

   a． ある1つの表記法変数または表記法定数。

   b． 表記法変数，表記法定数，言語の構文表示記号，およびキーワードの任意の集りを中括弧または大括弧でくくったもの。

4. 中括弧｛ ｝は，2つ以上の要素が構文上の単位を成していることを示すためのものである。たとえば，

   識別名　$\left\{\begin{matrix}\text{FIXED}\\ \text{FLOAT}\end{matrix}\right\}$

   構文上の単位がたてに積み重ねられている場合には，これらの中のいずれか1つを選ばなければならない。したがって，上の例は，「識別名」と言う変数の後にFIXEDまたはFLOATのどちらかを記入しなければならないことを示している。

5. 縦線 ｜ は，ある選択を行なうべきことを示すためのものである。たとえば，

   識別名　｛FIXED｜FLOAT｝

   この例は上の例と同じ意味を表わしている。本書においては，選択項目を示すのにこの双方の表記方法が使われている。

6. 大括弧〔　〕はオプションを示すためのものである。大括弧でくくられている項目に関しては，この項目を1つ記入してもよいし，この項目を記入しなくてもよい。なお，これ以外に，ある構文上の単位を他と区別するために大括弧を使うこともできる。たとえば，

（｛［下限：］上限｝｜＊）

この例は，星印あるいは変数「上限」を記入すべきことを示しているが，両方とも記入することはできない。「上限」を記入する場合には，必要に応じて，変数「下限」と特殊文字コロンとから成る構文上の単位を変数「上限」の前に記入することができる。

7. 3つの点・・・は，その直前にある構文上の単位を1つ以上続けて記入してもよいことを示すためのものである。たとえば，

〔数字〕・・・

変数「数字」は大括弧でくくられているから，これは記入してもしなくてもよい。これを記入する場合には，2つ以上続けて記入することもできる。

8. 下線は，言語の要素と表記法の要素とを混同する恐れがある場合にこの要素が言語の要素であることを示すためのものである。たとえば，

オペランド｛ε｜⊥｝ オペランド

これは，2つの変数「オペランド」の間に「および」(&)または「または」(|)を記入すべきことを示している。演算子|の下線は，これがPL/I言語の「または」を表わす記号であり，表記法の「または」を表わす記号でないことを示している。

B節： 文字セットおよびそのEBCDICとカード穿孔コード

60－文字セット

文　　字	カード穿孔コード	8-ビット EBCDIC コード
ブランク	無　穿　孔	0100 0000
.	12-8-3	0100 1011
<	12-8-4(12-8-6)	0100 1100
(	12-8-5(0-8-4)	0100 1101
+	12-8-6(12)	0100 1110
|	12-8-7(NA)	0100 1111
ε	12(NA)	0101 0000
$	11-8-3	0101 1011
*	11-8-4	0101 1100
)	11-8-5(12-8-4)	0101 1101
;	11-8-6	0101 1110
¬	11-8-7(NA)	0101 1111
-	11	0110 0000
/	0-1	0110 0001
,	0-8-3	0110 1011
%	0-8-4(NA)	0110 1100
—	0-8-5(NA)	0110 1101
>	0-8-6(8-6)	0110 1110
?	0-8-7(12-0)	0110 1111
:	8-2(8-5)	0111 1010
#	8-3(NA)	0111 1011
@	8-4(NA)	0111 1100
'	8-5(8-4)	0111 1101
=	8-6(8-3)	0111 1110
A	12-1	1100 0001
B	12-2	1100 0010
C	12-3	1100 0011
D	12-4	1100 0100
E	12-5	1100 0101
F	12-6	1100 0110
G	12-7	1100 0111
H	12-8	1100 1000
I	12-9	1100 1001
J	11-1	1101 0001
K	11-2	1101 0010
L	11-3	1101 0011
M	11-4	1101 0100
N	11-5	1101 0101
O	11-6	1101 0110
P	11-7	1101 0111
Q	11-8	1101 1000
R	11-9	1101 1001
S	0-2	1110 0010
T	0-3	1110 0011
U	0-4	1110 0100
V	0-5	1110 0101
W	0-6	1110 0110
X	0-7	1110 0111
Y	0-8	1110 1000
Z	0-9	1110 1001
0	0	1111 0000
1	1	1111 0001
2	2	1111 0010
3	3	1111 0011
4	4	1111 0100
5	5	1111 0101
6	6	1111 0110
7	7	1111 0111
8	8	1111 1000
9	9	1111 1001

複合記号	カード穿孔コード
<=	12-8-4,　8-6(12-8-6,8-3)
||	12-8-7,　12-8-7(NA)
**	11-8-4,　11-8-4
¬<	11-8-7,　12-8-4(NA)
¬>	11-8-7,　0-8-6(NA)
¬=	11-8-7,　8-6(NA)
>=	0-8-6,8-6(8-6,8-3)
/*	0-1,　11-8-4
*/	11-8-4,　0-1
->	11,　0-8-6(11,8-6)

括弧でくくられているカード穿孔コードは，対応するEBCDICコードとは異なる形をもつBCDICコードである。

NAは，BCDIC ではこの文字を表現することができないことを示している。BCDコンパイラー・オプションを指定してある場合に限り，PL/I 原始プログラムの中でBCDIC コードを使用することができる。ここには，BCDIC 8ビット・コードは示されていない。というのは，BCDオプションが指定されている場合，原始プログラムの中のBCDIC コードはすべてコンパイラーによってEBCDIC に変換されるからである。

PL/I 60－文字セットの中には，BCDIC で表現することができない文字が含まれている。しかし，48－文字セットの場合は，すべての文字をBCDIC で表現することができる（次ページを参照されたい）。したがって，BCDオプションと共に48－文字セット・オプションを指定すれば，PL/I のすべての操作を実行することができる。

48－文字セット

文　字	カード穿孔コード	8－ビット EBCDIC コード
ブランク	無 穿 孔	0100 0000
.	12－8－3	0100 1011
(	12－8－5（0－8－4）	0100 1101
+	12－8－6（12）	0100 1110
$	11－8－3	0101 1011
*	11－8－4	0101 1100
)	11－8－5（12－8－4）	0101 1101
－	11	0110 0000
／	0－1	0110 0001
,	0－8－3	0110 1011
'	8－5（8－4）	0111 1101
=	8－6（8－3）	0111 1110
A	12－1	1100 0001
B	12－2	1100 0010
C	12－3	1100 0011
D	12－4	1100 0100
E	12－5	1100 0101
F	12－6	1100 0110
G	12－7	1100 0111
H	12－8	1100 1000
I	12－9	1100 1001
J	11－1	1101 0001
K	11－2	1101 0010
L	11－3	1101 0011
M	11－4	1101 0100
N	11－5	1101 0101
O	11－6	1101 0110
P	11－7	1101 0111
Q	11－8	1101 1000
R	11－9	1101 1001
S	0－2	1110 0010
T	0－3	1110 0011
U	0－4	1110 0100
V	0－5	1110 0101
W	0－6	1110 0110
X	0－7	1110 0111
Y	0－8	1110 1000
Z	0－9	1110 1001
0	0	1111 0000
1	1	1111 0001
2	2	1111 0010
3	3	1111 0011
4	4	1111 0100
5	5	1111 0101
6	6	1111 0110
7	7	1111 0111
8	8	1111 1000
9	9	1111 1001

括弧でくくられているカード穿孔コードは，対応するEBCDICコードとは異なる形をもつ BCDICコードである。BCD コンパイラー・オプションを指定してある場合に限り，PL／I 原始プログラムの中で BCDIC コードを使用することができる。ここには BCDIC 8ビット・コードは示されていない。というのは，BCD オプションが指定されている場合，原始プログラムの中の BCDIC コードはすべてコンパイラーによって EBCDIC に変換されるからである。

複合記号	カード穿孔コード	60－文字セットの対応記号
..	12－8－3, 12－8－3	:
LE	11－3, 12－5	<=
CAT	12－3, 12－1, 0－3	||
**	11－8－4, 11－8－4	**
NL	11－5, 11－3	¬<
NG	11－5, 12－7	¬>
NE	11－5, 12－5	¬=
／／	0－1, 0－1	%
,.	0－8－3, 12－8－3	;
AND	12－1, 11－5, 12－4	ε
GE	12－7, 12－5	>=
GT	12－7, 0－3	>
LT	11－3, 0－3	<
NOT	11－5, 11－6, 0－3	¬
OR	11－6, 11－9	|
／*	0－1, 11－8－4	／*
*／	11－8－4, 0－1	*／
PT	11－7, 0－3	－>

注．　48－文字セットを使用する場合には，以下の諸規則を守らなければならない。

1. コロンを表わすためにピリオドを2つ記入する場合，この2つのピリオドの直前の文字がピリオドであるときはこのピリオドと後のピリオドとの間にブランクを1つ挿入しなければならない。

2. パーセント記号を表わすためにスラッシュを2つ記入する場合，その直前または直後の文字が星印であるときはそのスラッシュと星印との間にブランクを1つ挿入しなければならない。

3. 上に示したようにコンマとピリオドを続けて記入すると，これはセミコロンを表わすことになる。ただし，注釈，文字ストリング，またはピクチャー指定の中でコンマとピリオドが続けて記入されているとき，または，コンマとピリオドの直後に数字がある場合は，これはセミコロンを表わさない。

4. コンパイレーションの際に，48－文字セットを指定するためのコンパイラー・オプションが含まれていても，60－文字セットの記号を48－文字セットの記号と自由に混ぜて使用することができ，コンパイラーは60－文字セットの記号を有効な入力として受け入れる。

5. 48－文字セットの「予約」語（たとえば，GT，LE，CAT）の直前および直後にはブランクまたは注釈を置かなければならない。この規則を守らないと，コンパイラーがこれをどう解釈するか予知することができず，したがってコンパイラーによる解釈がユーザーの意図と合致しないこともある。

48文字セットの予約語の一部または全体を含んでいるレコードは，その長さが3文字以上でなければならない。

C節： キーワードとその省略形

キーワード	省　略　形	用　　　途
A〔(w)〕		形式項目
ABS(x)		組込み関数
ACOS(x)		組込み関数
%ACTIVATE	%ACT	プリプロセッサー・ステートメント
ADD(x_1, x_2, x_3〔, x_4〕)		組込み関数
ADDBUFF(n)		ENVIRONMENT 属性のオプション
ADDR(x)		組込み関数
ALIGNED		境界合せの属性
ALL(x)		組込み関数
ALLOCATE		ステートメント
ALLOCATION(x)		組込み関数
ANY(x)		組込み関数
AREA		条件
AREA〔(サイズ)〕		属性
ARGn		OPTIONS オプション／属性のオプション
ASCII		ENVIRONMENT 属性のオプション
ASIN(x)		組込み関数
ASSEMBLER	ASM	OPTIONS 属性のオプション
ASSOCIATE(ファイル名)		ENVIRONMENT 属性のオプション
ATAN(x_1〔, x_2〕)		組込み関数
ATAND(x_1〔, x_2〕)		組込み関数
ATANH(x)		組込み関数
AUTOMATIC	AUTO	記憶域クラスの属性
B〔(w)〕		形式項目
BACKWARDS		ファイルの属性
BASED〔(ロケーター式)〕		記憶域クラスの属性
BEGIN		ステートメント
BINARY	BIN	算術データ属性
BINARY(x_1〔, x_2〔, x_3〕〕)	BIN(x_1〔, x_2〔, x_3〕〕)	組込み関数
BIT〔(長さ)〕		ストリング属性
BIT(x_1〔, x_2〕)		組込み関数
BLKSIZE(ブロック・サイズ)		ENVIRONMENT 属性のオプション
BOOL(x_1, x_2, x_3)		組込み関数
BUFFERED	BUF	ファイル属性
BUFFERS({1|2})		ENVIRONMENT 属性のオプション
BUFOFF〔(n)〕		ENVIRONMENT 属性のオプション
BUILTIN		入口名の属性
BY		DO ステートメントのオプション
BY NAME		割当てステートメントのオプション
C(実数形式項目〔, 実数形式項目〕)		形式項目
CALL		ステートメントまたは INITIAL 属性のオプション
CEIL(x)		組込み関数
CHAR(x_1〔, x_2〕)		組込み関数
CHARACTER〔(長さ)〕	CHAR〔(長さ)〕	ストリング属性
CHECK〔(名前のリスト)〕		条件，条件接頭語
CLOSE		ステートメント
COBOL		ENVIRONMENT 属性または OPTIONS オプション／属性のオプション
COLBIN		ENVIRONMENT 属性のオプション
COLUMN(n)	COL(n)	形式項目
COMPLETION(x)	CPLN(x)	組込み関数，擬似変数
COMPLEX	CPLX	データの属性
COMPLEX(x_1, x_2)	CPLX(x_1, x_2)	組込み関数，擬似変数
CONDITION	COND	属性
CONDITION(名前)	COND(名前)	条件
CONJG(x)		組込み関数
CONNECTED	CONN	記憶域属性
CONSECUTIVE		ENVIRONMENT 属性のオプション
CONTROLLED	CTL	記憶域クラス属性
CONVERSION	CONV	条件，条件接頭語

キーワード	省略形	用途
COPY〔(ファイル式)〕		GETステートメントのオプション
COS(x)		組込み関数
COSD(x)		組込み関数
COSH(x)		組込み関数
COUNT(ファイル式)		組込み関数
CTLASA		ENVIRONMENT 属性のオプション
CTL360		ENVIRONMENT 属性のオプション
D		ENVIRONMENT 属性のオプション
DATA		GETまたはPUTステートメントのオプション
DATAFIELD		組込み関数
DATE		組込み関数
DB		ENVIRONMENT 属性のオプション
%DEACTIVATE	%DEACT	プリプロセッサー・ステートメント
DECIMAL	DEC	算術データ属性
DECIMAL(x_1〔,x_2〔,x_3〕〕)	DEC(x_1〔,x_2〔,x_3〕〕)	組込み関数
DECLARE	DCL	ステートメント
%DECLARE	%DCL	プリプロセッサー・ステートメント
DEFAULT	DFT	ステートメント
DEFINED	DEF	記憶域属性
DELAY(n)		ステートメント
DELETE		ステートメント
DESCRIPTORS		DEFAULTステートメントのオプション
DIM(x_1,x_2)		組込み関数
DIRECT		ファイル属性
DISPLAY		ステートメント
DIVIDE(x_1,x_2,x_3〔,x_4〕)		組込み関数
DO		ステートメント，GETまたはPUTステートメントの繰返し指定
%DO		プリプロセッサー・ステートメント
E(w,d〔,s〕)		形式項目
EDIT		GETまたはPUTステートメントのオプション
ELSE		IFステートメントのクローズ
%ELSE		%IFステートメントのクローズ
EMPTY		組込み関数
END		ステートメント
%END		プリプロセッサー・ステートメント
ENDFILE(ファイル式)		条件
ENDPAGE(ファイル式)		条件
ENTRY		属性またはステートメント
ENVIRONMENT	ENV	ファイルの属性またはCLOSEステートメントのオプション
ERF(x)		組込み関数
ERFC(x)		組込み関数
ERROR		条件
EVENT		READ, WRITE, REWRITE, DELETE, DISPLAY ステートメントのオプション，属性
EXP(x)		組込み関数
EXTENTNUMBER(n)		ENVIRONMENT 属性のオプション
EXTERNAL	EXT	有効範囲の属性
F(w〔,d〔,s〕〕)		形式項目
F		ENVIRONMENT 属性のオプション
FB		ENVIRONMENT 属性のオプション
FILE		属性
FILE(ファイル式)		入出力ステートメントのオプション
FINISH		条件
FIXED		算術データ属性
FIXED(x_1〔,x_2〔,x_3〕〕)		組込み関数
FIXEDOVERFLOW	FOFL	条件，条件接頭語
FLOAT		算術データ属性
FLOAT(x_1〔,x_2〕)		組込み関数
FLOOR(x)		組込み関数
FORMAT(形式リスト)		ステートメント
FORTRAN		OPTIONSオプション／属性のオプション
FREE		ステートメント
FROM(変数)		WRITEまたはREWRITEステートメントのオプション
FUNCTION		ENVIRONMENT 属性のオプション

キーワード	省略形	用途
GENERIC		入口名の属性
GENKEY		ENVIRONMENT属性のオプション
GET		ステートメント
GO TO	GOTO	ステートメント
%GO TO	%GOTO	プリプロセッサー・ステートメント
HBOUND(x_1, x_2)		組込み関数
HIGH(x)		組込み関数
HIGHINDEX		ENVIRONMENT 属性のオプション
IF		ステートメント
%IF		プリプロセッサー・ステートメント
IGNORE(n)		READ ステートメントのオプション
IMAG(x)		組込み関数，擬似変数
IN(エリア変数)		ALLOCATE および FREE ステートメントのオプション
%INCLUDE		プリプロセッサー・ステートメント
INDEX(x_1, x_2)		組込み関数
INDEXAREA (索引域のサイズ)		ENVIRONMENT 属性のオプション
INDEXD		ENVIRONMENT 属性のオプション
INDEXMULTIPLE		ENVIRONMENT 属性のオプション
INITIAL	INIT	記憶域属性
INPUT		ファイルの属性，OPEN ステートメントのオプション
INTER		OPTIONS オプション／属性のオプション
INTERNAL	INT	有効範囲の属性
INTO(変数)		READ ステートメントのオプション
KEY (ファイル式)		条件
KEY(x)		READ, DELETE, REWRITE ステートメントのオプション
KEYED		ファイルの属性
KEYFROM(x)		WRITE ステートメントのオプション
KEYLENGTH(n)		ENVIRONMENT 属性のオプション
KEYLOC(n)		ENVIRONMENT 属性のオプション
KEYTO (変数)		READ ステートメントのオプション
LABEL		属性
LENGTH(x)		組込み関数
LBOUND(x_1, x_2)		組込み関数
LEAVE		ENVIRONMENT 属性のオプション
LIKE		構造の属性
LINE(n)		形式項目，PUT ステートメントのオプション
LINENO(x)		組込み関数
LINESIZE		OPEN ステートメントのオプション
LIST		GET または PUT ステートメントのオプション
LOCATE		ステートメント
LOG(x)		組込み関数
LOG2(x)		組込み関数
LOG10(x)		組込み関数
LOW(x)		組込み関数
MAIN		OPTIONS オプションのオプション
MAX(x_1, x_2, .., xn)		組込み関数
MEDIUM		ENVIRONMENT属性のオプション
MIN(x_1, x_2, .., xn)		組込み関数
MOD(x_1, x_2)		組込み関数
MULTIPLY(x_1, x_2, x_3〔, x_4〕)		組込み関数
NAME (ファイル式)		条件
NOCHECK〔(名前のリスト)〕		条件接頭語
NOCONVERSION	NOCONV	条件接頭語
NOFIXEDOVERFLOW	NOFOFL	条件接頭語
NOLABEL		ENVIRONMENT 属性のオプション
NOMAP〔(引数リスト)〕		OPTIONSオプション／属性のオプション
NOMAPIN〔(引数リスト)〕		OPTIONSオプション／属性のオプション
NOMAPOUT〔(引数リスト)〕		OPTIONSオプション／属性のオプション
NOOVERFLOW	NOOFL	条件接頭語
NORESCAN		% ACTIVATEステートメントのオプション
NOSIZE		条件接頭語
NOSTRINGRANGE	NOSTRG	条件接頭語
NOSTRINGSIZE	NOSTRZ	条件接頭語
NOSUBSCRIPTRANGE	NOSUBRG	条件接頭語

キーワード	省略形	用途
NOTAPEMK		ENVIRONMENT 属性のオプション
NOUNDERFLOW	NOUFL	条件接頭語
NOWRITE		ENVIRONMENT 属性のオプション
NOZERODIVIDE	NOZDIV	条件接頭語
NULL		組込み関数
OFFSET〔(エリア名)〕		ロケーターの属性
OFFSET(x_1, x_2)		組込み関数
OFLTRACKS(n)		ENVIRONMENT 属性のオプション
OMR		ENVIRONMENT 属性のオプション
ON		ステートメント
ONCHAR		組込み関数, 擬似変数
ONCOUNT		組込み関数
ONCODE		組込み関数
ONFILE		組込み関数
ONKEY		組込み関数
ONLOC		組込み関数
ONSOURCE		組込み関数, 擬似変数
OPEN		ステートメント
OPTIONS(オプションのリスト)		入口名の属性, PROCEDUREおよびENTRYステートメントのオプション
ORDER		PROCEDUREおよびBEGINステートメントのオプション
OUTPUT		ファイル属性, OPENステートメントのオプション
OVERFLOW	OFL	条件, 条件接頭語
P'ピクチャー指定'		形式項目
PAGE		形式項目, PUTステートメントのオプション
%PAGE		リスティング制御ステートメント
PAGESIZE(w)		OPENステートメントのオプション
PASSWORD(合言葉指定)		ENVIRONMENT 属性のオプション
PICTURE	PIC	属性
POINTER	PTR	ロケーターの属性
POINTER(x_1, x_2)	PTR(x_1, x_2)	組込み関数
POLY(x_1, x_2)		組込み関数
POSITION(式)	POS(式)	記憶域の属性
PRECISION(x_1, x_2〔, x_3〕)	PREC(x_1, x_2〔, x_3〕)	組込み関数
PRINT		ファイル属性
PROCEDURE	PROC	ステートメント
%PROCEDURE	%PROC	プリプロセッサー・ステートメント
PROD(x)		組込み関数
PUT		ステートメント
R(x)		形式項目
RANGE		DEFAULTステートメントのオプション
RCE		ENVIRONMENT 属性のオプション
READ		ステートメント
REAL		データの属性
REAL(x)		組込み関数, 擬似変数
RECORD		ファイル属性
RECORD(ファイル式)		条件
RECSIZE(レコードの長さ)		ENVIRONMENT 属性のオプション
RECURSIVE		PROCEDURE ステートメントのオプション
REENTRANT		OPTIONS オプションのオプション
REFER		BASED 属性のオプション
REGIONAL({1|3})		ENVIRONMENT 属性のオプション
REORDER		PROCEDUREおよびBEGINステートメントのオプション
REPEAT(x_1, x_2)		組込み関数
REPLY(c)		DISPLAYステートメントのオプション
RESCAN		%ACTIVATE ステートメントのオプション
RETURN		ステートメント
RETURNS(属性のリスト)		入口名の属性, PROCEDUREステートメントのオプション
REVERT		ステートメント
REWRITE		ステートメント
ROUND(x_1, x_2)		組込み関数
SCALARVARYING		ENVIRONMENT 属性のオプション
SEQUENTIAL	SEQL	ファイルの属性

キーワード	省略形	用途
SET（ロケーター変数）		ALLOCATE, LOCATE, READ ステートメントのオプション
SIGN(x)		組込み関数
SIGNAL		ステートメント
SIN(x)		組込み関数
SIND(x)		組込み関数
SINH(x)		組込み関数
SIZE		条件，条件接頭語
SKIP〔(n)〕		形式項目，GETおよびPUTステートメントのオプション
%SKIP		リスト制御ステートメント
SNAP		ONステートメントのオプション
SQRT(x)		組込み関数
STACKER(x)		ENVIRONMENT属性のオプション
STATIC		記憶域クラスの属性
STATUS(x)		組込み関数，擬似変数
STOP		ステートメント
STREAM		ファイルの属性
STRING(x)		組込み関数，擬似変数
STRING（ストリング名）		GETおよびPUTステートメントのオプション
STRINGRANGE	STRG	条件，条件接頭語
STRINGSIZE	STRZ	条件，条件接頭語
iSUB		DEFINED属性の見かけ変数
SUBSCRIPTRANGE	SUBRG	条件，条件接頭語
SUBSTR(x_1, x_2〔, x_3〕)		組込み関数，擬似変数
SUM(x)		組込み関数
SYSIN		システム標準入力ファイルの名前
SYSPRINT		システム標準出力ファイルの名前
SYSTEM		ONまたはDECLAREステートメントのオプション
TAN(x)		組込み関数
TAND(x)		組込み関数
TANH(x)		組込み関数
THEN		IFステートメントのクローズ
%THEN		%IFステートメントのクローズ
TIME		組込み関数
TITLE（要素式）		OPENステートメントのオプション
TO		DO ステートメントのオプション
TRANSLATE(x_1, x_2〔, x_3〕)		組込み関数
TRANSMIT		条件
TRUNC(x)		組込み関数
U		ENVIRONMENT属性のオプション
UNALIGNED	UNAL	境界合せの属性
UNBUFFERED	UNBUF	ファイル属性
UNDEFINEDFILE（ファイル式）	UNDF（ファイル式）	条件
UNDERFLOW	UFL	条件，条件接頭語
UNLOAD		ENVIRONMENT属性のオプション
UNSPEC(x)		組込み関数，擬似変数
UPDATE		ファイルの属性
V		ENVIRONMENT属性のオプション
VALUE		DEFAULTステートメントのオプション
VARIABLE		ファイル，入口，またはラベル・変数識別名の属性
VARYING	VAR	ストリングの属性
VB		ENVIRONMENT属性のオプション
VERIFY		ENVIRONMENT属性のオプション
VERIFY(x_1, x_2)		組込み関数
VSAM		ENVIRONMENT属性のオプション
WAIT		ステートメント
WHEN（総称記述子のリスト）		GENERIC宣言の中のオプション
WHILE		DO ステートメントのオプション
WRITE		ステートメント
X(w)		形式項目
ZERODIVIDE	ZDIV	条件，条件接頭語

D節：　ピクチャー指定用の文字

ピクチャー指定用の文字は，PICTURE 属性または編集指示人出力のP形式項目の中で使用される。このいずれで用いられていても，ピクチャー指定用の個々の文字の意味に変りはない。ピクチャー指定の概念については，「第１３章：　編集およびストリングの取扱い」を参照されたい。

ピクチャー文字は，対応するデータ項目の属性について記述するのに使用される（このデータ項目は，ある変数の値であってもよいし，プログラムと外部記憶装置との間で転送されるデータ項目であってもよい）。

ピクチャー指定は，つねに，文字ストリング・データ項目または数字データ項目の文字表現について記述するのに用いられる。文字ストリング・ピクチャー項目は，英字，１０進数字，およびブランクから成るピクチャー項目である。数字ピクチャー項目は，そのデータ部分が１０進数字，１０進小数点，文字E，および，必要に応じて使用される正または負の符号から成るピクチャー項目である。数字ピクチャー項目においては，算術データと一緒に使用されることの多い他の文字（たとえば，通貨記号）も指定することができる。これらの文字は，数字変数の数字と一緒に貯えられ，その変数の文字ストリングとしての値の一部を成しているとみなされるけれども，その変数の算術データとしての値の一部を成しているとはみなされない。文字ストリング・ピクチャー項目および数字ピクチャー項目のいずれに関しても，許容される最大の文字数は１０２３である。また，数字ピクチャー項目において指定することのできる数字の桁数は，最高，１５である。

数字変数に割り当てられる算術データは，文字表現に変換される。数字データ項目に対しては編集（ゼロ消去や文字の挿入など）を指定することができるが，文字ストリング・ピクチャー・データに対して編集を指定することはできない。

数字ピクチャー指定をつけて宣言されている変数に割り当てられるデータ（または，数字ピクチャー形式項目のもとで書き出されるデータ）は，内部コード化算術データの形であるか，コード化算術データに変換し得るデータでなければならない。したがって，このような割当ての行なわれるデータには，数字および必要に応じて１０進小数点と符号しか含めることができない。編集用の文字（たとえば，通貨記号）もこれに含めてはならない。これに違反すると CONVERSION 条件が起こる。

P形式項目を用いて読み取られる数字データは，そのP形式項目の中の指定（編集用の文字を含む）に合致していなければならない。P形式項目において指示されている文字が入力の流れの中にないと，CONVERSION 条件が起こる。

文字ストリング・ピクチャー指定をつけて宣言されている変数に割り当てられるデータ（または，文字ストリング・ピクチャー形式項目のもとで書き出されるデータ）は，そのピクチャー指定とその１字１字が一致していなければならない（または，各々の文字がそのピクチャー指定に変換可能でなければならない）。これに違反すると，CONVERSION 条件が起こる。

P形式項目を用いて読み取られる文字ストリング・データは，そのP形式項目の中の指定に合致していなければならない。P形式項目の中で指定されている文字が入力の流れの中にないと，CONVERSION 条件が起こる。

この節に収録されているいくつかの図は，ピクチャー指定の相違によって，あるピクチャー変数に値を割り当てる際，あるいはP形式項目を使って値を印刷する際に，これらの値の表現がどう変って来るかを示すためのものである。各々の図には，データの当初の値，割当て前（または書出し前）の値を収容している変数の属性，ピクチャー指定，および数字ないしピクチャー文字ストリング変数の文字ストリングとしての値が示されている。

文字ストリング・データのためのピクチャー文字

文字ストリング・ピクチャー指定においては，次の３種類のピクチャー文字しか使用することができない。

X　　これは，使用している計算機が判別し得る内部的なビット構成をもつ任意の文字が，対応する位置に入り得ることを示すためのピクチャー文字である。

A　　これは，対応する位置に，任意の英字またはブランクが入り得ることを示すためのピクチャー文字である。

9　　これは，対応する位置に，任意の１０進数字またはブランクが入り得ることを示すためのピクチャー文字である。

ある１つの文字ストリング・ピクチャー指定には，AまたはXが少なくとも１つ含まれていなければならない。

D．１図は，文字ストリング・ピクチャー指定の例を示したものである。この図のbと言う文字はブランクを表わしている。割当て

原データの属性	原　デ　ー　タ （定数形式）	ピクチャー指定	文字ストリング値[1]
CHARACTER(5)	'9B/2L'	XXXXX	9B/2L
CHARACTER(5)	'9B/2L'	XXX	9B/
CHARACTER(5)	'9B/2L'	XXXXXXX	9B/2Lbb
CHARACTER(5)	'ABCDE'	AAAAA	ABCDE
CHARACTER(5)	'ABCDE'	AAAAAA	ABCDEb
CHARACTER(5)	'ABCDE'	AAA	ABC
CHARACTER(5)	'12/34'	99X99	12/34
CHARACTER(5)	'L26.7'	A99X9	L26.7

[1] 文字ストリング・ピクチャー指定により宣言されている変数は文字ストリング値しかとらない。

図D．1　文字ストリング・ピクチャー指定の例

は左寄せして行なわれ，必要に応じて右側にブランクが補充される点に注意されたい。

数字データのためのピクチャー文字

数字データは数値を表わしていなければならない。したがって，数字データと対応するピクチャー指定には，文字XやAが含まれていてはならない。数字データのためのピクチャー文字を用いて，そのデータの編集を詳細に指定することができる。

ある1つの数字変数は，その用途に応じて，次のような2種類の値をもつと考えることができる。

1. 算術値
2. 文字ストリング値

算術値とは，データ項目の10進数字，仮想上の10進小数点の位置，それに場合によって符号，によって表わされる値である。式の計算結果がコード化算術形式の値となるような式に数字変数が含まれている場合，またはコード化算術形式の変数，数字変数，ないしビット・ストリング変数に数字変数が割り当てられる場合には，つねに，その数字変数の算術値が使用される。このような場合には，数字変数の算術値は内部コード化算術形式の表現に変換される。

文字ストリング値とは，データ項目の10進数字ならびにピクチャー指定の中にあるすべての編集文字および挿入文字によって表わされる値である。ただし，ピクチャー文字Vによって指定される仮想上の小数点の位置は文字ストリング値に何の影響も及ぼさない。文字ストリング式の演算または文字ストリング変数への割当てにおいて数文字変数が使用されている場合，リスト指示出力ないしデータ指示出力のもとでデータが印刷される場合，または，数字変数に重ねて定義されている文字ストリング変数への参照が行なわれる場合には，つねに数字変数の文字ストリング値が使用される。このような場合には，データの変換は不要である。

数字指定のためのピクチャー文字は，次のように分類することができる。

- 数字指定文字と10進小数点指定文字
- ゼロ消去文字
- 挿入文字
- 符号・通貨記号
- 貸方記号，借方記号，上重ね穿孔符号
- 指数指定文字
- スケール因数

数字指定は1つ以上のフィールドから構成されており，各々のフィールドにおいてある1つの固定小数点数が表現される。したがって，浮動小数点数を表わす場合には，小数部を表わすためのフィールドと指数部を表わすためのフィールドが1つずつ使用されることになる。

原データの属性	原 デ ー タ （定数形式）	ピクチャー指定	文字ストリング値[1]
FIXED(5)	12345	99999	12345
FIXED(5)	12345	99999V	12345
FIXED(5)	12345	999V99	34500[2]
FIXED(5)	12345	V99999	00000[2]
FIXED(7)	1234567	99999	34567[2]
FIXED(3)	123	99999	00123
FIXED(5,2)	123.45	999V99	12345
FIXED(7,2)	12345.67	9V9	56[2]
FIXED(5,2)	123.45	99999	00123

[1]算術値は指定の中の数字と，Vの実際の位置ないし仮想位置とによって表わされる値である。

[2]この場合，PL／I のもとでは結果は不定である（左端の方の有効数字が切り捨てられているため）。 ここに示されている結果はシステム／360およびシステム／370のもとで得られるものである。SIZE 条件が解禁状態にあれば，この条件が提起される。

図D.2　数字ピクチャー指定の例

ピクチャー指定文字Vがあると，フィールドはサブフィールドに分割される。ある指定の途中にVがあるときは，このVの前にある部分と後にある部分は，それぞれこの指定のサブフィールドを成している。

数字データのピクチャー指定を行なうにあたって特に注意を要するのは，数字桁を指定するためのピクチャー文字が各フィールドに少なくとも1つはなければならないと言う点である。ただし，このピクチャー文字が数字指定文字9である必要はない。これ以外のピクチャー文字，たとえば，ゼロ消去文字（Zまたは＊またはY）を使って，数字桁を指定することもできる。ある1つの数字指定を定義するにあたっては，これらの文字を少なくとも1つ使用しなければならない。

数字指定文字と10進小数点指定文字

固定小数点10進数値を表わすための最も単純な形の数字指定では，ピクチャー文字9とVが使用される。

9　これは，データ項目の対応する桁に10進数字が入ることを指定するためのピクチャー文字である。

V　これは，対応するデータ項目の中のこの位置に10進小数点があると想定するよう指定するためのピクチャー文字である。ただし，この文字は，実際の10進小数点を挿入するよう指定するためのものではない。割り当てられた値の整数部と小数部分は，Vという文字に揃えられる。したがって，割り当てられた値の左右両端において切捨てが行なわれたり，ゼロが追加されたりすることがある。（左端の有効数字が切り捨てられた場合，その結果は予期し得ないものとなり，SIZE が解禁状態にあるとSIZE 割込みが起こる。） 固定小数点10進数値のピクチャー指定（または，浮動小数点10進数値のピクチャー指定の最初のフィールド）にVという文字がない場合には，そのフィールドの指定の右端にVがあるものと想定される。このため，必要に応じて，割り当てられた値の下位の桁が切り捨てられてこの値が整数値にされる。1つのピクチャー指定の中では，Vは1つしか使用することができない。

図D.2　は数字ピクチャー指定の例を示したものである。

ゼロ消去文字

ゼロ消去ピクチャー文字は文字ストリング値の数字桁を条件付で指定するためのものであり，この文字を使って先行ゼロを星印またはブランクで置き換えたり，先行ゼロでないゼロをブランクで置き換えたりすることができる。先行ゼロとは，固定小数点数の左端の方の桁ま

原データの属性	原　デ　ー　タ （定数形式）	ピクチャー指定	文字ストリング値[1]
FIXED(5)	12345	ZZZ99	12345
FIXED(5)	00100	ZZZ99	bb100
FIXED(5)	00100	ZZZZZ	bb100
FIXED(5)	00000	ZZZZZ	bbbbb
FIXED(5,2)	123.45	ZZZ99	bb123
FIXED(5,2)	001.23	ZZZV99	bb123
FIXED(5)	12345	ZZZV99	34500[2]
FIXED(5,2)	000.08	ZZZVZZ	bbb08
FIXED(5,2)	000.00	ZZZVZZ	bbbbb
FIXED(5)	00100	*****	**100
FIXED(5)	00000	*****	*****
FIXED(5,2)	000.01	***V**	***01
FIXED(5)	00100	YYYYY	bb1bb
FIXED(5)	10203	9Y9Y9	1b2b3
FIXED(5,2)	000.04	YYYVY9	bbbb4

[1] 算術値は指定の中の数字とVの実際の位置ないし仮想位置によって表わされる値である。

[2] この場合，PL／I のもとでは結果は不定である（左端の方の有効数字が切り捨てられているため）。　ここに示した結果はシステム／360およびシステム／370のもとで得られるものである。SIZE 条件が解禁状態にあれば，この条件が提起される。

図D.3　ゼロ消去の例

たは浮動小数点数を構成している2つの部分の各々の左端の方の桁にあるゼロで，10進小数点の仮想位置の左側にあり，しかもより高位の桁に1～9の範囲の数字がないものを言う。ある数の一番左側にあるゼロ以外の数字およびこれより右側にあるすべての数字（ゼロとゼロ以外の数字）が有効数字である。浮動小数点数に関しては，指数フィールドに先行ゼロがある場合もあると言う点に注意されたい。

Z　これは数字桁を条件付で指定するためのピクチャー文字であり，この文字を指定すると，データの対応する桁にある先行ゼロがブランクと置き換えられる。データの対応する桁に先行ゼロがなければ，この桁の数字はブランクと置き換えられない。ピクチャー文字Zは，ピクチャー文字＊あるいは浮動ピクチャー文字と同じフィールドにあってはならない。また，同一フィールドの中のピクチャー文字9，T，I，R，またはYの右側にピクチャー文字Zを記入することもできない。

＊　これは数字桁を条件付で指定するためのピクチャー文字である。この文字はピクチャー文字Zと同様にして使用されるが，この文字を指定した場合には先行ゼロが星印と置き換えられる。ピクチャー文字＊は，ピクチャー文字Zまたは浮動ピクチャー文字と同一のサブフィールドにあってはならない。また，同一フィールドの中のピクチャー文字9，T，I，R，またはYの右側にピクチャー文字＊を記入することもできない。

Y　これは数字桁を条件付で指定するためのピクチャー文字であり，この文字を指定することによって，（先行ゼロであると否とを問わず）対応する桁にあるゼロがブランクと置き換えられる。対応する桁がゼロでない場合，この桁の数字がブランクと置き換えられることはない。

図D.3は，ゼロ消去文字の使用例を示したものである。この図

の中のｂと言う文字はブランクを表わしている。

注：　ピクチャー文字Ｖの右側にピクチャー文字Ｚまたは＊を記入する場合には，この指定の小数部分のすべての桁だけでなく，整数部のすべての桁にもそれぞれピクチャー文字Ｚまたは＊を記入しなければならない。ピクチャー文字Ｖの右側のすべての桁にゼロ消去文字が記入されていても，この値の小数部分のゼロが消去されるのは，小数部分のすべての桁がゼロでありしかも整数部のすべての桁が消去されている場合だけである。このような場合，このデータ項目の文字ストリング値はブランクまたは星印だけである。小数部分に有効数字が１つでも含まれている場合には，小数部分の桁の数字はブランクや星印では置き換えられない。

挿入文字

ピクチャー文字コンマ（，），小数点（．），スラッシュ（／），およびブランク（Ｂ）は，挿入文字と呼ばれるものである。これらの文字を指定すると，指定された文字が数字データまたは文字データの対応する位置に挿入される。これらの文字は，数または文字の位置を指示するためのものではなく，数字と数字または文字と文字の間に挿入される。ただし，実際には，各々の挿入文字は，消去されると否とを問わず，文字ストリング値の中のある１つの文字の位置を表わしている。コンマ，小数点，およびスラッシュは条件付き挿入文字である。すなわち，ゼロ消去文字のストリングの中でこれらの文字が使用されている場合，これらの文字も消去されることがある。ブランク（Ｂ）は無条件挿入文字である。この文字は，つねに，対応する位置にブランクを挿入するよう指定するものである。

注：　挿入文字は文字ストリング値の場合にだけ用いられる。これらの文字はデータ項目の算術値に関しては何の指定も行なわない。

，　この文字を指定すると，ゼロ消去が行なわれない場合に，数字データの対応する位置にコンマが挿入される。ゼロ消去が行なわれる場合，コンマが挿入されるのは，コンマの入るべき位置の左側に消去されない数字があるとき，または，コンマの入るべき位置のすぐ左にＶがありしかも小数部分に有効数字があるとき，または，コンマがストリングの最初にあるとき，または，コンマの前に数字桁を指定するのでない文字があるとき，だけである。上のいずれにも該当しない場合にゼロ消去が行なわれている場合はすべて，コンマ挿入文字は，あたかもこの文字がコンマの入るべき位置の直前にあるゼロ消去文字であるかのように取り扱われる。

．　これは，コンマ・ピクチャー文字と同じように使用される。ただし，この文字を指定した場合，対応する位置には小数点（．）が入る。この文字を指定しても，固定小数点１０進数のピクチャー指定においてこの位置にその数の小数点が合わされることはない。この文字はデータ項目の算術値の一部を成しているわけではない。数の小数点をある位置に合わせるという機能をもっているのはピクチャー文字Ｖだけである。実際にＶが記入されていない場合には，フィールド内の左端の桁の右に小数点があるものと想定され，この想定に従って数の小数点の位置が決められる。小数点挿入文字がどこか他の位置にあってもこの間の事情に変りはない。Ｖを記入しても小数点が印刷されるわけではないから，Ｖとの関係を考えて小数点（またはコンマまたはスラッシュ）を記入すると，固定小数点数（または浮動小数点数）の整数部が終り小数部分が始まる場所に小数点（またはコンマまたはスラッシュ）を挿入することができる。印刷をするときは，通常，このような措置が必要である。このような場合，小数点はＶの直前または直後に置かなければならない。小数点がＶの直前にある場合，Ｖの左側に消去されない数字があるときに限って小数点が挿入される。たとえ，小数部分のすべての桁に有効数字があっても，この点に変りはない。Ｖの直後に小数点がある場合には，Ｖの右側の数字がすべて消去されると小数点も消去される。ただし，小数部分に有効数字が１つでもあると，小数点が（小数点と有効数字との間に介在するゼロと共に）挿入される。

／　これは，コンマ・ピクチャー文字と同じように使用される。ただし，この文字を指定した場合，対応する位置にスラッシュ（／）が挿入される。

Ｂ　これは，数字データの文字ストリング値の対応する位置につねにブランクを挿入するよう指定するためのピクチャー文字である。

図Ｄ．４　は，挿入文字の使用例を示したものである。この図の中のｂという文字はブランクを表わしている。

符号・通貨記号

ピクチャー文字Ｓ，＋，および－は，数字データの中の符号を指定するためのものである。ピクチャー文字＄は数字データの文字ストリング値の中の通貨記号を指定するためのものである。

これらのピクチャー文字を使用するにあたっては，これを固定的に用いることもできるし浮動的に用いることもできる。固定的な使

原データの属性	原データ（定数形式）	ピクチャー指定	文字ストリング値[1]
FIXED(4)	1234	9,999	1,234
FIXED(6,2)	1234.56	9,999V.99	1,234.56
FIXED(4,2)	12.34	ZZ.VZZ	12.34
FIXED(4,2)	00.03	ZZ.VZZ	bbb03
FIXED(4,2)	00.03	ZZV.ZZ	bb.03
FIXED(4,2)	12.34	ZZV.ZZ	12.34
FIXED(4,2)	00.00	ZZV.ZZ	bbbbb
FIXED(9,2)	1234567.89	9,999,999.V99	1,234,567.89
FIXED(7,2)	12345.67	**,999V.99	12,345.67
FIXED(7,2)	00123.45	**,999V.99	***123.45
FIXED(9,2)	1234567.89	9.999.999V,99	1.234.567,89
FIXED(6)	123456	99/99/99	12/34/56
FIXED(6)	123456	99.9/99.9	12.3/45.6
FIXED(6)	001234	ZZ/ZZ/ZZ	bbb12/34
FIXED(6)	000012	ZZ/ZZ/ZZ	bbbbbb12
FIXED(6)	000000	ZZ/ZZ/ZZ	bbbbbbbb
FIXED(6)	000000	**/**/**	********
FIXED(6)	000000	**B**B**	**b**b**
FIXED(6)	123456	99B99B99	12b34b56
FIXED(3)	123	9BB9BB9	1bb2bb3
FIXED(2)	12	9BB/9BB	1bb/2bb

[1] 算術値は指定の中の数字とVの実際の位置ないし仮想位置によって表わされる値である。

図D.4　挿入文字の使用例

用法とは，対応する桁に符号，通貨記号，またはブランクを必ず挿入するよう指定するものである。浮動的使用法とは，先行ゼロを消去するよう指定するものである。この場合，この種のピクチャー文字と対応する桁で消去されているものの右端の桁に符号，ブランク，または通貨記号が挿入される。（ただし，すべての桁に浮動ピクチャー文字が記入されていても，このデータ項目の値がゼロである場合には，浮動ピクチャー文字は挿入されない。）

浮動ピクチャー文字を指定するには，1つのピクチャー・フィールドの中にこの文字を2つ以上記入すればよい。すなわち，ある1つのフィールドの中に通貨記号（$）が1つ記入されている場合には，これは固定的使用法であると解釈され，ある1つのフィールドの中に通貨記号が2つ以上記入されている場合には，これは浮動的使用法であると解釈される。浮動ピクチャー文字を指定するときは，この文字が挿入される可能性のあるすべての桁にこの文字を記入しなければならない。

浮動ピクチャー文字は，ストリングの形式になっていなければならない。ストリングとは同一の浮動ピクチャー文字の列である。さらに必要に応じて，ストリングの中にVと挿入文字コンマ，小数点，スラッシュ，Bの中のいずれか1つとを記入することができる。ストリングの中または直後にある挿入文字スラッシュ，コンマ，または小数点は浮動ピクチャー文字ストリングの一部であるとみなされる。文字Bを記入すると，この文字がどの位置にあっても，必ず

その位置にブランクが挿入される。データ項目の算術値がゼロの場合を除き，Vが記入されているところで浮動ピクチャー文字ストリングが終了する。データ項目の算術値がゼロの場合には，Vは無視される。1つのピクチャー指定フィールドの中では，浮動ピクチャー文字ストリングは1つしか記入することができない。浮動ピクチャー文字ストリングの左側に数字桁があってはならない。また，ピクチャー文字＊あるいはZと浮動ピクチャー文字ストリングとを同一のフィールドに記入してはならない。

浮動ピクチャー文字ストリングの中の文字スラッシュ，コンマ，および小数点と対応するデータ内の位置には，次のいずれか1つが挿入される。

- 左側に有効数字がある場合には，スラッシュ，コンマ，または小数点。
- 右隣りの桁にこのフィールドの最高位の有効数字がある場合には，浮動記号。
- このフィールドの最高位の有効数字が右に2桁以上離れたところにある場合には，ブランク。

浮動ピクチャー文字ストリングの中に浮動ピクチャー文字が<u>n</u>個含まれているとすると，このストリングは<u>n</u>－1個の条件付き数字桁と対応する。左端の浮動ピクチャー文字と対応する桁には浮動ピクチャー文字ないしブランクが挿入されるだけであり，この桁に数字が挿入されることはない。同一のフィールドの中で2種類のピクチャー文字をいっしょに浮動的に使用することはできない。

浮動ピクチャー文字ストリングの中にVという文字が含まれている場合，このVはその先行部分を1つのサブフィールドとして区切る働きをし，Vの右側のサブフィールドにあるすべての桁もまた，この2番目のサブフィールドを開始する浮動ピクチャー文字ストリングの一部になっていなければならない。

それぞれのフィールドには，1種類の符号文字しか記入することができない。固定文字としてS，＋，または－を使用する場合には，浮動小数点数指定では小数部フィールドならびに指数フィールドの中の全数字桁の左側にこの文字を記入することができるし，また固定小数点数指定では，全数字桁の右ないし左にこの文字を記入することができる。

Vの右にある数字桁すべてに浮動ピクチャー文字が記入されている場合，このサブフィールドの中で消去が行なわれるのは整数部および小数部分の桁がすべてゼロのときだけである。このような場合，編集された後のデータ項目は全体がブランクになる（ただし，このフィールドの最初のところに挿入文字が入ることはある）。小数部分に1つでも有効数字がある場合には，小数部分全体が何の消去も受けずに残される。

$　これは通貨記号を指定するためのピクチャー文字である。この文字が2つ以上記入されている場合にはこれは浮動ピクチャー文字である。それ以外の場合は，これは固定文字である。固定文字とは，対応する桁にこの文字を挿入するよう指定するためのものである。固定文字として$を指定する場合には，指定のフィールドの中にある全数字桁の左側にこの文字を記入するか，指定の全数字桁の右側にこの文字を記入するかしなければならない。この文字の浮動的使用法についての詳細は，上記の説明を参照されたい。

S　これは，データの値が≧0の場合には正符号（＋）を指定し，これ以外の場合には負符号（－）を指定するためのピクチャー文字である。この文字を固定的に使用することもできるし，浮動的に使用することもできる。この際適用される規則は，通貨記号に適用される規則と同じである。

＋　これは，データの値が≧0の場合には正符号（＋）を指定し，これ以外の場合にはブランクを指定するためのピクチャー文字である。この文字を浮動的に使用することもできるし，固定的に使用することもできる。この際適用される規則は，通貨記号に適用される規則と同じである。

－　これは，データの値が＜0の場合には負符号（－）を指定し，これ以外の場合にはブランクを指定するためのピクチャー文字である。この文字を浮動的に使用することもできるし，固定的に使用することもできる。この際適用される規則は，通貨記号に適用される規則と同じである。

ピクチャーへの割当てが行なわれる前，またはピクチャーへの割当てが行なわれている間に10進数の小数部分が切り捨てられ，このために最終的な値がゼロになった場合，切捨て前のこの10進数の値の符号がピクチャーに挿入される。したがって，このピクチャーの中の符号はこの10進数値の算出過程によって左右されることもある。

図D．5　は，浮動ピクチャー文字の使用例を示したものである。この図の中のbと言う文字はブランクを表わしている。

貸方記号，借方記号，上重ね穿孔符号

一対の文字からなるCR（貸方）とDB（借方）は，実数の数字データ項目の符号を指定するためのものであり，通常，商業分野の報告書で使用される。

ピクチャー文字T，I，Rは，いずれも，数字データの中の対応する数字桁に上重ね穿孔符号をつけるよう指定するためのものである。上重ね穿孔符号とは，ある数字と同じカード桁に穿孔された12-穿孔（正）または11-穿孔（負）のことである。これはこの算術データ項目の符号を表わすためのものである。固定小数点数の指定においては，上重ね穿孔符号を1つ使用することができるだけである。浮動小数点数の指定においては，小数部フィールドおよび指数フィールドに各々1つずつ，合計2つの上重ね穿孔符号を使用することができる。なお，上重ね穿孔文字はフィールドの中のどの数字桁にで

原データの属性	原データ（定数形式）	ピクチャー指定	文字ストリング値[1]
FIXED(5,2)	123.45	$999V.99	$123.45
FIXED(5,2)	012.00	99$	12$
FIXED(5,2)	001.23	$ZZZV.99	$bb1.23
FIXED(5,2)	000.00	$ZZZV.ZZ	bbbbbbb
FIXED(1)	0	$$$.$$	bbbbbb
FIXED(5,2)	123.45	$$$9V.99	$123.45
FIXED(5,2)	001.23	$$$9V.99	bb$1.23
FIXED(2)	12	$$$,999	bbb$012
FIXED(4)	1234	$$$,999	b$1,234
FIXED(5,2)	2.45	SZZZV.99	+bb2.45
FIXED(5)	214	SS,SS9	+214
FIXED(5)	−4	SS,SS9	−4
FIXED(5,2)	−123.45	+999V.99	b123.45
FIXED(5,2)	−123.45	−999V.99	−123.45
FIXED(5,2)	123.45	999V.99S	123.45+
FIXED(5,2)	001.23	++B+9V.99	bbb+1.23
FIXED(5,2)	001.23	−−−9V.99	bbb1.23
FIXED(5,2)	−001.23	SSS9V.99	bb−1.23

[1] 算術値は指定の中の数字とVの実際の位置ないし仮想位置によって表わされる値である。

図D.5　浮動ピクチャー文字の使用例

も指定することができる。この指定があると，指定されている数字桁の数に上重ね穿孔符号がつく。

CR　これは，データの値がゼロより小さい場合に，対応する位置にCRという文字を挿入するよう指定するためのピクチャー文字である。これ以外の場合には，対応する位置に2つのブランクが入る。CRという文字の指定は，フィールドの中の全数字桁の右側にしか記入することができない。

DB　これは，CRと同様にして使用される。ただし，この場合には，対応する位置にDBという文字が挿入される。

T　この文字は，入力のとき，対応する桁の数字に，データの値が≧0ならば12−穿孔が上重ね穿孔されており，データの値が＜0ならば11−穿孔が上重ね穿孔されているということを指定するものである。また，これはこの上重ね穿孔が文字ストリング値で表示されることも表わしている。

I　この文字は，入力のとき，対応する桁の数字に，データの値が≧0の場合12−穿孔が上重ね穿孔されており，これ以外の場合，この桁には何の上重ね穿孔もされていないことを指定するものである。また，この文字は，データの値が≧0ならば文字ストリング値で上重ね穿孔が表示されることを指定する働きももつ。

R　この文字は，入力のとき，対応する桁の数字に，データの値が＜0の場合11−穿孔が上重ね穿孔されることを指定するものである。これ以外の場合，この桁には上重ね穿孔は何も行なわれない。また，この文字は，データの値が＜0ならば文字ストリング値で上重ね穿孔が表示されることを指定する働きももつ。

原データの属性	原データ（定数形式）	ピクチャー指定	文字ストリング値[1]
FIXED(3)	−123	$Z.99CR	$1.23CR
FIXED(4,2)	12.34	$ZZV.99CR	$12.34bb
FIXED(4,2)	−12.34	$ZZV.99DB	$12.34DB
FIXED(4,2)	12.34	$ZZV.99DB	$12.34bb
FIXED(4)	1021	999I	102A
FIXED(4)	−1021	Z99R	102J
FIXED(4)	1021	99T9	10B1

[1] 算術値は指定の中の数字とVの実際の位置ないし仮想位置によって表わされる値である。

図D.6　ピクチャー文字CR，DB，T，I，R の使用例

原データの属性	原データ（定数形式）	ピクチャー指定	文字ストリング値[1]
FLOAT(5)	.12345E06	V.99999F99	.12345E06
FLOAT(5)	.12345E−06	V.99999ES99	.12345E−06
FLOAT(5)	.12345E+06	V.99999KS99	.12345+06
FLOAT(5)	−123.45E+12	S999V.99ES99	−123.45E+12
FLOAT(5)	001.23E−01	SSS9.V99ESS9	+123.00Eb−3
FLOAT(5)	001.23E+04	ZZZV.99KS99	123.00+02
FLOAT(5)	001.23E+04	SZ99V.99ES99	+123.00E+02
FLOAT(5)	001.23E+04	SSSSV.99E−99	+123.00Eb02

[1] 算術値は，小数部で表わされる値に 10^n を掛けた積である（ただし，nは指数フィールドで示されている値）。

図D.7　浮動小数点数ピクチャー指定の例

図D.6はCR，DB，および上重ね穿孔文字の使用例を示したものである。

注：同一フィールド内で，ピクチャー文字CR，DB，T，I，Rとこれら以外の符号文字とを一緒に使用することはできない。

指数指定文字

ピクチャー文字KおよびEは，ある浮動小数点10進数について記述している数字指定の指数フィールドの境を指定するためのものである。指数フィールドは，つねに，数字浮動小数点ピクチャー指定の最後のフィールドである。同一のフィールドの中でピクチャー文字KとEを共に使用することはできない。

K　これは，対応する桁の右側が指数フィールドであることを指定するためのピクチャー文字である。これは，数字データ項目の中の文字を指定するためのものではない。

E　これは，対応する桁にEという文字が入り，このEが指数フィールドの始まりを表わしていることを指定するためのピクチャー文字である。

原データの属性	原データ（定数形式）	ピクチャー指定	文字ストリング値[1]
FIXED(4,0) FIXED(7,0) FIXED(5,5) FIXED(6,6)	1200 −1234500 .00012 .012345	99F(2) S999V99F(4) 99F(−5) 999V99F(−4)	12 −12345 12 12345
[1] 算術値は文字ストリング値に10^nをかけた積である（ただし，nはスケール因数）。			

図D.8　スケール因数ピクチャー文字の使用例

文字ストリング値の場合は，最初のフィールド（小数部）の最高位の有効数字が，数字浮動小数点ピクチャー指定の最初の数字桁指定文字と対応する桁に来るよう（それがゼロ消去文字であっても)，指数の値が調整される。

図D.7 は指数指定文字の使用例を示したものである。この図の中のbという文字はブランクを表わしている。

スケール因数

ピクチャー文字Fは固定小数点10進数のスケール因数を指定するためのものである。この文字を指定するときは，ピクチャー指定の右端に次のような形式で記入する。

F （〔＋１－〕 10進整数定数）

F　これは，括弧でくくられている10進整数定数（必要に応じて符号がつけられている）がスケール因数であることを指定するためのピクチャー文字である。スケール因数とは，この変数の算術値の10進小数点が，文字ストリング値における10進小数点の仮想位置よりもスケール因数の値に相当する桁数分だけ右（スケール因数が正の場合）または左（スケール因数が負の場合）にあることを指定するためのものである。

スケール因数が－128～127の範囲を越える位取りを暗示していてはならない。

図D.8 は，スケール因数ピクチャー文字の使用例を示したものである。

E節： 編集指示の形式項目

この節では，GETまたはPUTステートメントの形式リストの中に記入することのできる編集指示形式項目の各々について説明する。

形式項目には，データ形式項目，制御形式項目，および遠隔形式項目の３種類がある。

この節においては，この３種類について別々に説明し，各々の説明が済んだところで各種類に属する形式項目を列挙する。さらにその後で，アルファベット順に各形式項目について詳しく説明する。

データ形式項目

データ形式項目とは，ある１つのデータ項目が外部媒体上でとる形式について記述するためのものである。

入力の際には，流れの中のデータは連続した１つの文字のストリングとみなされる。流れの中にあっては，ブランクおよび引用符はすべて文字として取り扱われる。GETステートメントの各々のデータ形式項目によって，流れから読み取るべき文字の桁数が指定され，これらの文字の解釈方法を指示される。この際，ストリングを引用符で囲んではならない。また，ビット・ストリングを識別するのにBという文字を使ってはならない。指示されている解釈方法で流れの中の文字を解釈することができない場合には，CONVERSION条件が起こる。

出力の際には，流れの中のデータは形式リストによって指定されている形式をとる。PUTステートメントの中の各々のデータ形式項目によって，対応するデータ項目が文字形式で入れられるフィールドの幅が指定され，またこの値がとるべき形式が指示される。ストリングを囲むための引用符も挿入されないし，ビット・ストリングを識別するためのBという文字も挿入されない。

出力の流れの中のデータ項目を１つ１つ区切るために先行ブランクが自動的に挿入されるわけではない。ストリング・データは，指定された幅をもつフィールドの中で左寄せされるが，算術データは右寄せされる。算術データを文字タイプに変換する場合の規則では，（先行ゼロと置き換わるブランクのほかに）最高３つの先行ブランクが挿入される可能性があるため，交換されたフィールドの算術項目には通常少なくとも１つの先行ブランクができることになる。ただし，指定されているフィールド幅に先行ブランクを収容する余裕がなければ，流れの中に先行ブランクは現われない。指定されているフィールド幅が充分でないために起こる切捨ては，算術項目の場合には左端で行なわれ，ストリング項目の場合には右端で行なわれる。

編集指示転送においては，入力の際にも出力の際にも２進データの値はつねに１０進数形式で表示される点に注意されたい。

以下にデータ形式項目を列挙する。

固定小数点形式項目	F
浮動小数点形式項目	E
複素数形式項目	C
ピクチャー形式項目	P
ビット・ストリング形式項目	B
文字ストリング形式項目	A

制御形式項目

制御形式項目は，あるファイルと結びつけられているデータ・セットのレイアウトを指定するためのものである。以上に制御形式項目を列挙する。

ページ形式項目	PAGE
印刷行スキップ形式項目	SKIP
印刷行位置形式項目	LINE
印刷桁位置形式項目	COLUMN
スペース形式項目	X

制御形式項目は，データ・リストの処理が終了する前に出て来なければ何の効果ももたない。

PAGEおよびLINE両形式項目は，出力だけに関係するものであり出力の中でもPRINT属性をもつファイルにだけ適用され

る。これに対して，形式項目SKIP，COLUMN，およびXは，入力と出力の双方に適用される。

形式項目PAGE，SKIP，およびLINEは，PUT ステートメント（SKIP に関しては，さらにGETステートメント）のオプションとしてこれらを指定したのと同じ効果をもつ。ただし，形式項目としてこれらを指定した場合には，この形式項目が形式リストの中に現われたとき初めてその効果が現われるのに対して，オプションとしてこれらを指定すると，データの転送が行なわれる前にその効果が現われる。

COLUMN 形式項目は，現在の印刷行または次の印刷行の指定された文字位置にファイルの位置設定を行なうためのものである。GET STRINGまたはPUT STRING ステートメントではCOLUMN 形式項目を使用することができない場合もある点に注意されたい。

スペース形式項目は水平方向の相対的なスペースを指定するためのものである。入力の際には，この形式項目で，流れの中のスキップすべきまたは無視すべき文字の数を指定する。出力の際には，この形式項目で，流れの中に挿入すべきブランクの数を指定する。

あるファイルを開いた後このファイルに対して出される最初のGETまたはPUTステートメントの中で制御形式項目を指定した場合の効果については，「J節： ステートメント」のOPENステートメントの項を参照されたい。

遠隔形式項目

遠隔形式項目は，この遠隔形式項目と置き換えられる形式リストを含んでいるFORMATステートメントのラベルを指定するためのものである。遠隔形式項目は次のような形をとる。

R（ステートメント・ラベル標識）

「ステートメント・ラベル標識」は，ラベル定数またはスカラー・ラベル変数である。

形式項目の用法

以下のリストに挙げられている形式項目は，大部分，その後に指定が続くようになっている。ピクチャー形式項目および遠隔形式項目を除くすべての形式項目に関しては，この指定の中に含まれている式は，10進整数定数，要素変数，または他の要素式として与えることができる。後の方のあるデータ項目に結びつけられる形式項目の式の中で，入力操作中に変数に割り当てられる値を使用することができる。形式項目が使用されるたびに，その式の値が計算され，整数に変換される。

形式項目のリスト（アルファベット順）

A 形式項目

A形式項目は次のような形式をとる。

A［（フィールドの幅）］

文字ストリング形式項目は，ある文字ストリングが外部媒体上でとる表現形式について記述するためのものである。

一般規則：

1. 「フィールドの幅」は，この形式項目が使用されるたびに，その値が計算され，整数に変換される式である（変換された後の整数が負であってはならない）。この式で，データの流れの中にあってこのストリングを収容している（または収容することになっている）フィールドの文字の桁数を指定する。

2. 入力の際，ここで指定されている桁数分の文字が，データの流れから読み取られ，必要に応じて変換，切捨て，またはパディングを経た後データ・リストの中の対応する要素に割り当てられる。入力の際には，必ずフィールドの幅を指定しなければならず，この値がゼロのときにはこのストリングはナル・ストリングであるとみなされる。データの流れの中に引用符があった場合，これはストリングの中の文字であるものとして取り扱われる。

3. 出力の際，データ・リスト内の対応する要素が，必要に応じて，文字ストリングに変換され，指定されているフィールドの幅に合わせて切捨てないし右端へのブランクの追加が行なわれたあとでデータの流れの中に入れられる。フィールドの幅がゼロの場合には，この形式項目およびデータ・リスト内のこれと対応する要素はスキップされ，データの流れに文字が入れられることはない。なお，ストリングの両端を囲むための引用符はデータの流れには出されない。ここでフィールドの幅が指定されていない場合には，データ・リスト内の指名されている要素の文字ストリングとしての長さがフィールドの幅であるものとみなされる（ただし，この要素は,必要に応じて，「F節： データの変換と式の値の計算」に示されている規則に準じた変換を行なった後のものである）。

B 形式項目

B形式項目は次のような形式をとる。

B［（フィールドの幅）］

ビット・ストリング形式項目は，あるビット・ストリングが外部媒体上でとる表現形式について記述するためのものである。

各ビットは文字の0か1で表わされる。

一般規則：

1. 「フィールドの幅」は，この形式項目が使用されるたびにその値が計算され，整数に変換される式である（変換された後の値が負であってはならない）。この式で，このビット・ストリングを収容している（または収容することになっている）データの流れの文字の桁数を指定する。

2. 入力の際には，このビット・ストリングの文字表現は，ここで指定されているフィールド内の不特定の場所に入っている。フィールド内のビット・ストリングの前後にブランクがあることが多いが，ブランクは無視される。データ・リストの中の対応する要素にこのビット・ストリングが割り当てられるとき，必要に応じて，変換が行なわれる。入力の際には，必ず，フィールドの幅を指定しなければならない。フィールドの幅の値がゼロの場合，このビット・ストリングはナル・ストリングであるものとみなされる。ストリングの中に0および1以外の文字（たとえば，ストリング内部のブランク，引用符，あるいはB）が含まれている場合，CONVERSION 条件が起こる。

3. 出力の際には，このビット・ストリングの文字表現は，ここで指定されているフィールドに左寄せされ，必要に応じて右端で切捨てないしブランクの追加が行なわれる。必要ならば，さらに，ビット・ストリングへの変換も行なわれる。引用符も識別用文字Bも挿入されない。フィールドの幅がゼロの場合，この形式項目とデータ・リスト内のこれと対応する要素はスキップされ，データの流れには文字は何も置かれない。フィールドの幅が指定されていない場合，データ・リスト内の指名されている要素のビット・ストリングとしての長さがフィールドの幅であるとみなされる（ただし，この要素は，必要に応じて，「F節：　データの変換と式の値の計算」に示されている規則に準じた変換を行なった後のものである）。

C　形式項目

C形式項目は次のような形式をとる。

C（実数形式項目［，実数形式項目］）

複素数形式項目は，ある複素数データ項目が外部媒体上でとる表現形式について記述するためのものである。

一般規則：

1. 各「実数形式項目」のところで，F形式項目，E形式項目，またはP形式項目のいずれか1つを指定する。P形式項目は，数字データについて記述しているものでなければならない。P形式項目で文字ストリング・データについて記述してはならない。

2. 入力の場合には，複素数形式項目は，データの流れ内の隣接する2つのフィールドにある複素数形式項目の実部と虚部について記述する。2番目の実数形式項目を省略した場合，2番目の実数形式項目は最初の実数形式項目と同じであるものとみなされる。文字Iを記入すると，CONVERSION条件が起こる。

3. 出力の場合には，実数形式項目は，複素数データ項目の実部と虚部がデータの流れの中でとる形について記述する。2番目の実数形式項目を省略した場合，2番目の実数形式項目は最初の実数形式項目と同じであるとみなされる。複素数の虚部にIという文字が挿入されるわけではない。2番目の実数形式項目（1つだけ指定されているときは，最初の実数形式項目）がF形式項目かE形式項目の場合は，虚部の値がゼロより小さいときにだけ内部符号が印刷される。実数形式項目がP形式項目の場合は，Sまたは－または＋というピクチャー文字が指定されているときに限り，符号が印刷される。虚部にIという文字を挿入したいときは，データ・リストの中でこの複素数データ項目を指定している変数の直後に別個のデータ項目として，Iという文字を指定しなければならない。この場合，このIに対応する形式項目（A形式項目かP形式項目）を指定しなければならない。

COLUMN　形式項目

COLUMN形式項目は次のような形式をとる。

COLUMN（文字の位置）

印刷桁位置形式項目は，現在の印刷行または次の印刷行上の指定の文字位置に，ファイルの位置設定を行なうためのものである。入力，出力のどちらのファイルに対してもこの形式項目を使用することができる。

一般規則：

1. 「文字の位置」は，この形式項目が使用されるたびにその値が計算され，整数に変換される式である（変換された後の整数が負であってはならない）。

2. 現在の行の上で，ここで指定されている文字位置をまだ過ぎていない場合には，ファイルは，現在の行上のこの文字位置に位置設定される。入力のときは，この文字位置までの間にある文字位置は無視される。出力のときは，ここで指定されている文字位置までの間にある文字位置はブランクになる。ファイルがすでにここで指定されている文字位置の後に位置設定されている場合は，現在の行の終りまで

行き，新しい行が開始される。したがって，このような場合には，この形式項目は次の行に適用されることになる。

3. ここで指定されている文字位置が現在の行の右端の文字位置よりも右にある場合，または，文字位置として指定されている式の値がゼロの場合には，文字位置は１であるものとみなされる。

注： 右端の文字位置は，下記の要因によって決まる。

a． 出力ファイルに関しては，右端の文字位置は印刷行の桁数によって決まる。

b． 入力ファイルに関しては，コンパイラーが現在の論理レコードの長さをもとにして行の長さ，したがって，右端の文字位置を決める。V形式のレコードの場合には，こうして決定される行の長さは論理レコード長から制御情報を収容しているバイトの数を引いて得られた値である。

4. データ・リストの処理が終了する前にCOLUMN 形式項目が出て来ないと，この形式項目は何の効力ももたない。

5. GET STRINGまたはPUT STRING ステートメントでは，COLUMN 形式項目を使用してはならない。

E 形式項目

E形式項目は次のような形式をとる。

E（フィールドの幅，小数部分の桁数，
［，有効数字の桁数］）

浮動小数点形式項目は，浮動小数点形式の１０進算術データが外部媒体上でとる表現形式について記述するためのものである。

一般規則：

1. 「フィールドの幅」，「小数部分の桁数」，および「有効数字の桁数」として式を指定してもよい。この形式項目が使用されるたびに，この式の値が計算されこの値が整数に変換される。

「フィールドの幅」と記されているところで，このフィールドの中の文字の総数を指定する。

「小数部分の桁数」と記されているところで，小数部の中の１０進小数点より右側にある桁数を指定する。「有効数字の桁数」と記されているところで，小数部として必要な桁数を指定する。

2. 入力のときには，データの流れの中のデータ項目は，ここで指定されているフィールドの中の任意の場所にある，符号付の（任意選択）１０進浮動小数点定数または１０進固定小数点定数の文字表現である。このデータ項目が固定小数点数の場合，指数はゼロであると想定される。

浮動小数点数は外部媒体上で次のような形をとる。

［＋｜－］小数部 ⎛⎧［E］｛＋｜－｝⎫ 指数⎞
⎝⎩ E ［＋｜－］⎭ ⎠

小数部は１０進固定小数点定数でなければならない。

a． 数は，指定されているフィールドの中のどの場所にあってもよい。フィールド内の数の前後にはブランクがあるのが普通であるが，このブランクは無視される。ただし，フィールド全体がブランクである場合には，CONVERSION条件が起こる。１０進小数点を使用しない場合には，小数部分の桁数を表わす式で，小数部内の仮想上の小数点の右側にある文字位置の数を指定する。数の中で１０進小数点が使用されている場合には，この小数点の方が有効であり，形式項目で指定されている小数部分の桁数は無効となる。

「フィールドの幅」で指定する値には，数の後に置かれるブランク，指数部の桁，（正または負の符号を指定している場合には）符号の桁，（Eという文字を指定している場合には）Eのための桁，（小数部で１０進小数点を使用している場合には）１０進小数点の桁なども含まれる。

b． 指数は１０進整数定数である。指数と指数の前の符号ないしEという文字を省略した場合には，指数はゼロであると想定される。

3. 出力の際には，内部データが浮動小数点数の形に変換され，外部媒体上のここで指定されているフィールドには次のような一般形式をとるデータ項目が入る。

［－］ ｛s－dの桁数｝・｛dの桁数｝
E｛＋｜－｝ 指数

ここで，Sは有効数字の桁数であり，dは小数部分の桁数である。値は必要に応じてまるめられる。

a． 指数は２桁の１０進整数定数であり，２桁ともゼロ

であってもよい。小数部の最初の桁がゼロ以外の数字になるよう，指数は自動的に調整される。値がゼロの場合には，１０進小数点の左側にあるすべての数字桁（ただし，１０進小数点のすぐ左の桁を除く）に対してゼロ消去が行なわれる。これ以外のすべての数字桁にはゼロが入る。

b． 上に示した形式の数を収容しても出力として指定されているフィールドに余裕がある場合には，この数は右寄せされ，数の左側にブランクをつけて拡張される。有効数字の桁数が指定されていない場合，小数部分の桁数に１を加えた数がその桁数であるとみなされる。データ項目の値が負でない場合は，そのフィールドの幅は有効数字の桁数に５を加えた数またはそれ以上でなければならない。データ項目の値が負の場合，そのフィールドの幅は有効数字の桁数に６を加えた数またはそれ以上でなければならない。ただし，小数部分の桁数がゼロの場合には，１０進小数点は書き出されず，したがってフィールドの幅として上に示した数字が各々１だけ小さくなる。

c． 計算機内部のデータのまるめは次のようにして行なわれる。切捨ての結果右端の桁が１つ失なわれ，さらにこの桁の数字が５以上のときは，切捨ての行なわれた桁の左の桁に１が加えられる。

b． フィールドの幅として指定されている値が不適当であるために有効数字が失なわれたり，符号が失なわれたりすると，SIZE条件が起こる。

F　形式項目

F形式項目は次のような形式をとる。

F（フィールドの幅［，小数部分の桁数
［，スケール因数］］）

固定小数点形式項目は，固定小数点数形式の１０進算術データ項目が外部媒体上でとる表現形式について記述するためのものである。

一般規則：

1. 「フィールドの幅，」「小数部分の桁数，」および「スケール因数」として要素式を指定してもよい。要素式を指定した場合，この形式項目が使用されるとき要素式の値が計算され，計算された値が整数に変換される。こうして計算されたフィールドの幅および小数部分の桁数は，どちらも負であってはならない。

2. 入力の際には，データの流れの中のデータ項目は，ここで指定されているフィールドの中の任意の場所にある１０進固定小数点定数（必要に応じて，符号がつけられている）の文字表現となっている。フィールド内の数の前後にはブランクがあるのが普通であるが，このブランクは無視される。フィールド全体がブランクである場合には，ゼロであると解釈される。

   小数部分の桁数を指定しない場合は，ゼロであるとみなされる。

   スケール因数が指定されておらず，しかもフィールド内に１０進小数点がない場合には，小数部分の桁数として指定されている式は，フィールド内の仮想１０進小数点より右側にある桁数を示すことになる。データの中に実際に１０進小数点がある場合には，この１０進小数点が有効であり，小数部分の桁数として指定されている式は無効になる。

   スケール因数が指定されている場合，データの流れの中のデータ項目の値は，事実上１０の<u>p</u>乗倍されたものになる（<u>p</u>はスケール因数の整数値）。したがって，<u>p</u>が正であれば，１０進小数点が右に<u>p</u>桁ずれたところにあるものとしてこの数が取り扱われる。<u>p</u>が負であれば，１０進小数点が左に<u>p</u>桁ずれたところにあるものとしてこの数が取り扱われる。ただし，ここで基準としている１０進小数点の位置は，フィールド内に１０進小数点がある場合にはこれが置かれている場所であり，フィールド内に１０進小数点がない場合には小数部分の桁数として指定されている式によって指示される場所である。

3. 出力の際には，計算機内部のデータが，必要に応じて，固定小数点数に変換される。外部媒体上のデータは，必要に応じてまるめられ，指定のフィールドに右寄せされて入れられている文字表現の１０進固定小数点数である。

   この形式項目の中でフィールドの幅だけが指定されている場合には，この数の整数部分しか書き出されない。１０進小数点も書き出されない。

   フィールドの幅も小数部分の桁数も指定されているが，スケール因数は指定されていないという場合には，この数の整数部分も小数部分も共に書き出される。このような場合，小数部分の桁数として指定された値（<u>d</u>）がゼロより大きければ右端から<u>d</u>桁目の左に１０進小数点が挿入される。また，小数部分の桁数が<u>d</u>より小さい場合には，数字桁の後にゼロがつけ加えられる（<u>d</u>はフィールドの幅より小さくなければならない）。

１０進小数点の左側にあるすべての数字桁（ただし，１０進小数点のすぐ左の桁を除く）に対しては，先行ゼロの消去が行なわれる。

計算機内部のデータのまるめは，次のようにして行なわれる。切捨ての結果右端の桁が１つ失なわれ，さらにこの桁の数字が５以上の場合，切り捨てられた桁の左の桁に１が加えられる。

スケール因数が指定されている場合（スケール因数の整数値をp とする），データ・リストの中の対応する要素の値は，事実上，１０のp 乗倍されてから外部媒体上でとるべき文字表現に編集される。また，小数部分の桁数がゼロの場合には，この数の整数部分だけが書き出される。

出力の際，ここで指定されている固定小数点数の値がゼロより小さい場合には，外部媒体上のこの数の文字表現の左に負符号がつく。固定小数点数がゼロ以上の場合には，符号はつかない。したがって，固定小数点数の値が負の場合には，フィールドの幅として指定された値は符号および10進小数点のための文字位置を含んでいなければならない。

フィールドの幅が充分でないために文字が失われるようなことがあると，ＳＩＺＥ条件が起こる。

LINE 形式項目

ＬＩＮＥ形式項目は次のような形式をとる。

ＬＩＮＥ（印刷行番号）

印刷行位置形式項目は，ＰＲＩＮＴファイルの現在のページ または次のページにおいて次のデータ項目を印刷すべき印刷行を指定するためのものである。

一般規則：

1. 「印刷行番号」として式を指定することができる。式を指定した場合，この形式項目が使用されるたびにこの式の値が計算され，計算された値が整数に変換される（変換された後の整数が負であってはならない）。

2. 必要に応じて，ブランク行が挿入される。

3. ここで指定されている印刷行番号が現在の印刷行番号以下である場合，または，OPENステートメントのPAGESIZEオプション（ないし省略時オプション）で定められている限界を越えるものである場合は，ENDPAGE条件が起こる。ただし，ここで指定されている印刷行番号が現在の印刷行番号に等しく，その１桁目の文字がまだ転送されていない場合には，ＳＫＩＰ（０）項目と同様の措置がとられる（すなわち，行送りをせずに紙送り機構が戻る）。

4. 「印刷行番号」がゼロのときは，この番号が１であるとみなされる。

5. データ・リストの処理が終了する前にＬＩＮＥ形式項目が出て来ないと，この形式項目は何の効力ももたない。

P 形式項目

Ｐ形式項目は次のような形式をとる。

Ｐ（ピクチャー指定）

ピクチャー形式項目は，数字データおよび文字ストリング・データが外部媒体上でとる表現形式について記述するためのものである。

「ピクチャー指定」について詳しくは，「Ｄ節： ピクチャー指定用の文字」および「Ｉ節： 属性」のＰＩＣＴＵＲＥ属性の項を参照されたい。

入力の際には，ピクチャー指定は，データの流れの中のデータ項目の形について記述する働きをし，さらに，数字指定に関しては データ項目の算術値の解釈方法について指示する働きをする。ピクチャー指定では，入力の流れの中のデータについて，編集文字によって表現されている文字も含めて正確に記述しなければならないという点に注意されたい。ピクチャー指定で指示されている文字が入力の流れの中に現われないと，ＣＯＮＶＥＲＳＩＯＮ 条件が起こる。

出力の際には，データ・リストの中の対応する要素の値は，ピクチャー指定で指定されている形に変換されてから，データの流れに書き出される。

PAGE 形式項目

ＰＡＧＥ形式項目は次のような形式をとる。

ＰＡＧＥ

ページ形式項目は，新しいページを開始するよう指定するためのものである。この形式項目は，ＰＲＩＮＴ ファイルに対してだけ使用することができる。

一般規則：

1. 新しいページを開始するということは，次のページの最初の印刷行にファイルを位置設定することを意味する。

2. データ・リストの処理が終了する前にＰＡＧＥ形式項目が出て来ないと，この形式項目は何の効力ももたない。

R 形式項目

R形式項目は次のような形式をとる。

R（ステートメント・ラベル標識）

遠隔形式項目を使用すると，FORMAT ステートメントの形式項目によって，この遠隔形式項目を置き換えることができる。

一般規則：

1. 「ステートメント・ラベル標識」は，FORMATステートメントのステートメント・ラベルをその値とするラベルを表わすラベル定数，要素ラベル変数，または関数参照である。なお，このFORMATステートメントには形式リストが含まれており，これがR形式項目と置き換えられる。

2. R形式項目とR形式項目で指定されているFORMAT ステートメントとは，同じブロックの内部になければならない。（さらに，これらが回帰的に実行されるプロシージャーの中にある場合には，これらは同一の呼出しで実行されなければならない。）

3. FORMATステートメントの内部に回帰があってはならない。すなわち，遠隔FORMATステートメントの中に，ステートメント・ラベル標識としてそのステートメント自体を指名しているR形式項目があってはならないし，このR形式項目が，当初のFORMATステートメントの指名につながるような別の遠隔FORMATステートメントを指名してはならない。このような回帰を避けるには，遠隔形式項目の中で参照されているFORMATステートメントにさらに別の遠隔形式項目を書き入れることのないよう注意すればよい。

4. GETまたはPUTステートメントを実行するとき解禁状態にある条件は，R形式項目で参照される遠隔FORMATステートメントの実行時にも解禁状態になければならない。

5. オン・ユニットがGETステートメントまたはPUT ステートメントだけで構成されているときには，このステートメントの中で遠隔形式項目を使うことはできない。

SKIP 形式項目

SKIP形式項目は次のような形式をとる。

SKIP ［（次の行の相対的な位置）］

行スキップ形式項目は新しい行を開始するよう指定するためのものである。

一般規則：

1. 「次の行の相対的な位置」として要素式を指定することができる。要素式を指定した場合，この形式項目が使用されるたびに要素式の値が計算され，計算された値が整数に変換される（変換された後の整数が負であってはならない）。PRINTファイル以外のファイルを対象としている場合は，この値はゼロより大きくなければならない。これがゼロ以下であったり，これを省略したりした場合には，この値が１であるとみなされる。

2. 現在の行からここで指定されている行数だけ先の行が新しい行となる。

3. 「次の行の相対的な位置」の値が１より大きい場合，入力のときは１行以上の行が無視され，出力のときは１行以上のブランク行が挿入される。

4. PRINTファイルを対象としている場合に限り，「次の行の相対的な位置」の値がゼロでもよい。値がゼロの場合は，行送りをせずに紙送り機構が最初の印刷桁に戻る。したがって，すでに文字が印刷されていると，その部分には文字が重複して印刷されることになる。

5. PRINTファイルの場合には，ここで指定されている次の行の相対的な位置がOPENステートメントのPAGESIZEオプション（または，省略時オプション）によって定められている限界を越えていると，ENDPAGE条件が起こる。

6. データ・リストの処理が完了する前にSKIP 形式項目が出て来ないと，この形式項目は何の効力ももたない。

X 形式項目

X形式項目は次のような形式をとる。

X（フィールドの幅）

スペース形式項目は，データの流れの中のデータ項目とデータ項目との間隔を指示するためのものである。PRINT ファイル以外のファイルに対してもこの形式項目を使用することができる。

一般規則：

1. 「フィールドの幅」は，この形式項目が使用されるたびにその値が計算され，計算された値が整数に変換される式である（変換された後の整数が負であってはならない）。この整数は，データの流れの中の現在位置から次のフィールドまでのブランクの数を指定するものである。

2. 入力のときは，データの流れの中で，指定の桁数分の文字が飛び越され，これらの文字はプログラムに転送されない。

3. 出力のときは，指定の桁数分のブランク文字がデータの流れに挿入される。

4. データ・リストの処理が終了する前にスペース形式項目が出て来ないと，この形式項目は何の効力ももたない。

優先順位	演 算 子	演算のタイプ	参 照 先	注
1	＊＊	算術	図F. 4	結果はコード化算術形式
	接頭演算子＋，−	算術	−	オペランドがコード化算術形式の場合変換は不要
			目標フィールド：FIXED DECIMAL	オペランドが固定小数点１０進数のCHARACTERストリング表現ないし数字（PICTURE）表現をとっている場合オペランドはFIXED DECIMALに変換される
			目標フィールド：FLOAT DECIMAL	オペランドが浮動小数点１０進数の数字（PICTURE）表現をとっている場合オペランドはFLOAT DECIMALに変換される。
			目標フィールド：FIXED BINARY	オペランドがBITストリングの場合オペランドはFIXED BINARYに変換される
	¬	ビット・ストリング	目標フィールド：BIT	BIT以外のデータはすべてBITに変換される
2	＊／	算術	図F. 4	結果はコード化算術形式
3	挿入演算子＋，−	算術	図F. 4	結果はコード化算術形式
4	||	連結	目標フィールド：CHARACTER	一方のオペランドまたは双方のオペランドがCHARACTERないしDECIMALの場合，CHARACTERでないオペランドがCHARACTERに変換される
			目標フィールド：BIT	オペランドがBITとBINARYの場合または双方のオペランドがBINARYの場合，BITでないオペランドがBITに変換される
5	< ¬< <= = ¬= >= > ¬>	比較	図F. 5	結果はつねに'1'Bか'0'Bである
6	&	ビット・ストリング	目標フィールド：BIT	BIT以外のデータはすべてBITに変換される
7			ビット・ストリング	目標フィールド：BIT	BIT以外のデータはすべてBITに変換される
		すべての形の割当て	個々の目標フィールドを参照のこと	すべての形の割当ての目標フィールドの属性を図F. 3に示す。

図F. 1　演算の優先順位と変換規則の手引

演算子は，優先順位に従ってあげてある。したがって，第1グループが優先順位の一番上のものであり，第7グループが一番下のものである。同一の優先順位の欄にあげられている演算子はすべて同じ優先順位のものである。たとえば，指数演算子＊＊の優先順位は，接頭演算子＋および−や「not」演算子¬の優先順位と同じである。

1つの式の中に優先順位1の演算子が2つ以上ある場合は，式の中でのこれらの演算子の位置が右側のものほど，その優先順位が上である。したがって，一番右側にある指数演算子または接頭演算子＋ないし−または¬演算子が一番上の優先順位をもつことになる。以下，順に，左に行くほど優先順位は下になる。これらの演算子以外のすべての演算子に関しては，1つの式の中に同じ優先順位の演算子が2つ以上ある場合，その相互の優先順位は式の中の演算子の位置が左側にあるほどその演算子の優先順位は上になる。

CEIL値の表

１０進数と２進数の間で変換が行なわれる場合には，精度属性の値を３．３２で除算ないし乗算し，この結果を整数にまるめる必要がある。右の表はこの除算ないし乗算とまるめの後の結果を示したものである。

原フィールドの精度属性が正である場合には，乗算のあと積の小数点以下の端数を切り上げて得られる整数（PL／I の用語では，CEIL値）がとられる。たとえば，

5＊3.32＝16.6

CEIL(16.6)＝17

したがって，新しい精度属性は１７である。

この最終結果は右の表から直接読み取ることもできる。すなわち，CEIL(n＊3.32)の欄のn＝5のところを見ると，１７と記されている。

原フィールドの精度属性が負の場合（精度属性のスケール因数はおそらく負）は，結果の絶対値の小数点以下の端数を切り上げ，これに本来の符号をつけて得られる整数（PL／Iの用語では，FLOOR値）がとられる。たとえば，

(−5)＊3.32＝(−16.6)

FLOOR(−16.6)＝(−17)

したがって，新しいスケール因数は−１７である。

この例から正の値のみならず負の値の場合にも右の表を利用できることがわかるであろう。すなわち，

FLOOR(−n)＝−CEIL(n)

n	CEIL(n＊3.32)	n	CEIL(n／3.32)
1	4	1−3	1
2	7	4−6	2
3	10	7−9	3
4	14	10−13	4
5	17	14−16	5
6	20	17−19	6
7	24	20−23	7
8	27	24−26	8
9	30	27−29	9
10	34	30−33	10
11	37	34−36	11
12	40	37−39	12
13	44	40−43	13
14	47	44−46	14
15	50	47−49	15
16	54	50−53	16
17	57	54−56	17
18	60	57−59	18
19	64	60−63	19
20	67	64−66	20
21	70	67−69	21
22	74	70−73	22
23	77	74−76	23
24	80	77−79	24
25	83	80−82	25
26	87	83−86	26
27	90	87−89	27
28	93	90−92	28
29	97	93−96	29
30	100	97−99	30
31	103	100−102	31
32	107	103−106	32
33	110	107−109	33
		110−112	34
		113−116	35

図F.2　CEIL(n＊3.32)とCEIL(n／3.32)の値

以下の事例はいずれかの属性への変換を引き起こす：

事例	目標フィールドの属性
割当て	割当て記号の左側にある変数の属性
式のオペランド	式の値を計算する規則に従って決められる
流れ入力（GETステートメント）	受取り側フィールドの属性
流れ出力（PUTステートメント）	編集指示の流れである場合形式リストによって決められる これ以外の場合は文字ストリング
PROCEDUREまたはENTRYへの引数	対応するパラメーターの属性
組込み関数または擬似変数への引数	関数または擬似変数によって決まる
INITIAL属性	初期値化される変数の他の属性
RETURNステートメントの式	PROCEDUREまたはENTRYステートメントで指定されている属性
DOステートメント，BYまたはTOオプション	制御変数の属性

以下のステートメントは文字ストリングへの変換を引き起こす：

ステートメント	オプション	ストリングの長さの最大値
DISPLAY		原フィールド，最大72文字
レコード入出力	KEYFROM	指定されているキーの長さ＋8
	KEY	指定されているキーの長さ＋8
OPEN	TITLE	変換規則に従って決められる。最大8文字

以下のステートメントは2進整数への変換を引き起こす：

ステートメント	オプション／属性／参照	精度
DECLARE／ALLOCATE／DEFAULT	長さ，大きさ	15
	次元，限界，	15
	繰返し因数	15
DELAY	ミリセカンド	31
FORMAT	反復因数	15
（およびGETとPUTの形式項目）	w	15
	d	7
	s	7
	p	7
OPEN	LINESIZE	15
	PAGESIZE	15
I／O	SKIP	15
	LINE	15
	IGNORE	15
WAITステートメント	式のオプション	31
ステートメント一般	添字	15

図F.3　変換を引き起こす条件

F節： データの変換と式の値の計算

この節の目的は，ユーザーが，混成式（問題処理データも含めて）を分析して，個々の混成式においてどのような変換が行なわれるかを判断し，このような変換が式の最終結果にどのような影響を及ぼすかを見る上での便宜を提供することである。ここでは，割当てを混成式の特別の例であるとみなすことにする。ある式が次のいずれかに該当するとき，この式は「混成」式と呼ばれる。

1. オペランドの属性が同一でない。
   たとえば，

   DCL A CHAR(6), B FIXED BINARY(31);

   A=B;

   この場合，Bは，Aに割り当てられる前に，文字ストリングの形に変換される。

2. 行なわれる演算に適合しない属性をもつオペランドがある。
   たとえば，

   DCL C BIT(10) VARYING;

   C=C+C;

   この場合，加算が行なわれる前にCは算術値に変換される。さらに，この加算の和（算術値）は，Cに再び割り当てられる前に，ビット・ストリングの形に変換される。

この節では，このような変換を引き起こす条件ならびに変換規則をすべて取り上げる。

割当てや式の値の計算以外の場合にも，変換が行なわれることがある。このような場合にも，この節に示す規則を直接適用することができる。図F.3は変換を引き起こすような条件をすべて列挙したものである。

説明の順序

変換規則については，図F.1，図F.4，図F.5，および原フィールド・目標フィールド変換規則の項で説明されている。

図F.1は，実施される演算を列挙して，式の値を計算するにあたっての演算の優先順位を示したものである。この図には，さらに，図F.4および図F.5を参照する際の指針が載っており，どの原フィールド・目標フィールド変換規則を見ればより詳細な説明が得られるかも示されている。

図F.4は，すべての算術演算について各々どのような変換が行なわれるかを示したものである。

図F.5は，問題処理データの比較のすべてについて，各々どのような変換が行なわれるかを示したものである。

以下にあげるデータ・タイプの各々について，原フィールド・目標フィールド変換規則を示す。

コード化算術形式：

FIXED BINARY
FIXED DECIMAL
FLOAT BINARY
FLOAT DECIMAL

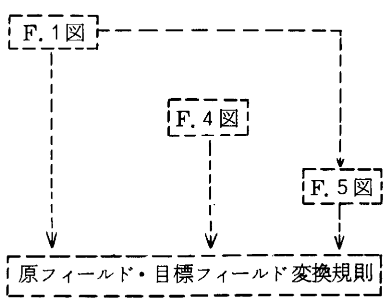
PICTURE（数字）

CHARACTER

BIT

以下のページでは，上記のデータ・タイプをそれぞれ目標フィールドのデータとして取り上げ，他のデータ・タイプのデータが原フィールドにある場合の変換規則をすべて示してある。演算子が含まれていない変換であっても，つねに，以下に示す原フィールド・目標フィールド変換規則を直接適用することができる。この点については，図F.3を参照されたい。

次の図は，図F.1，図F.4，図F.5，および原フィールド・目標フィールド変換規則の相互関係を示したものである。



これらの外に図F.2があるが，これは変換後の精度を新たに計算する際の手助けをしようとするものである。

この節の図の配置は，ある特定のページを読みながら，関係する

情報の大部分に目を通すことができるように工夫されている。つまり，図F.1と図F.2は一緒に本文とは別の折込みページに収録されており，図F.4と図F.5はそれぞれこの節の末尾の折込みページに収録されている。したがって，図F.1，図F.2，図F.4または図F.5，それに特定の原フィールド・目標フィールド変換規則の四者に同時に目を通すことができる。

変換規則の使用例

次の例は，図および原フィールド・目標フィールド変換規則の使用方法を示すためのものである。

```
DCL FB FIXED BINARY(5),
    FD FIXED DECIMAL(5,2),
    CH CHARACTER(12);
CH=4.2*FB+FD;
```

図F.1に示されている優先順位から，この割当てステートメントは次の3ステップに分けて実行されることがわかる。

1. 結果1 ＝ 4．2＊FB
2. 結果2 ＝結果1＋FD
3. CH＝結果2

各ステップの結果の属性を求める方法を次に説明する。

ステップ1

定数4.2に対して，FIXED DECIMAL(2,1)という属性が暗示される。

図F.4 aを見て，この定数の属性を第1オペランドの属性とし，FBの属性を第2オペランドの属性とする欄を探す。すると，5という参照コードが得られる。図F.4 cの参照コード5の欄を見ると，この結果の属性がFIXED BINARY(p,q)であることがわかる。この乗算の精度は次のように計算する。

p＝1＋(1＋2＊3.32)＋5＝14
q＝(1＊3.32)＋0＝4

- 結果1の属性はFIXED BINARY(14,4)である。

ステップ2

まず，図F.4 aを使う。結果1の属性を第1オペランドの属性とし，FDの属性を第2オペランドの属性とする欄を探すと7という参照コードが得られる。図F.4 cの参照コード7の欄を見ると，結果の属性がFIXED BINARY(p,q)であることがわかる。この加算の精度は次のようにして求める。

p＝1+MAX(14−4,((1+5＊3.32)−2＊3.32))+7＝19
q＝MAX(4,(2＊3.32))=7

- 結果2の属性はFIXED BINARY(19,7)である。

ステップ3

原フィールド・目標フィールド変換規則の中のFIXED BINARYからCHARACTERへの変換規則を見る。原フィールドのデータは，まず，FIXED DECIMAL(p,q)に変換される。ここで，

p＝1＋19／3.32＝7
q＝7／3.32＝3

原フィールドがFIXED DECIMALであり，目標フィールドがCHARACTER である場合の変換規則を調べる。すると，この10進定数がひとまず長さ10のストリングに割り当てられることがわかる。

このストリングがCHに割り当てられ，CHの右端にブランク文字が2つ補充される。

コード化算術形式

FIXED BINARY, FIXED DECIMAL, FLOAT BINARY, FLOAT DECIMALの4つのデータ・タイプはすべて，コード化算術データである。これらのデータ・タイプの間で行なわれる変換の規則は，目標フィールドとして各々のデータ・タイプを取り上げてあるところに示されている。しかし，次の一般的な注意事項を心に留めておいていただきたい。

- 10進数を2進数に変換する場合，および，固定小数点10進数と浮動小数点10進数との間で変換を行なう場合には，右端の桁の切捨てのために値に小さな差が生ずることがある。

- 複素数値を実数値に変換する場合には，虚数部が無視される。実数値を複素数値に変換する場合には，虚数部はゼロになる。

PICTURE（数字）

データは，まず，10進数であると解釈され，対応するPICTURE 指定をもとにしてスケールと精度が決定される。この後，このデータ項目は目標フィールドの基数，スケール，モード，および精度に変換される。目標フィールドとしてコード化算術データを取り上げているページで，原フィールドとしてFIXED DECIMALまたはFLOAT DECIMALを使用する項目を参照のこと。

CHARACTER

原フィールドのストリングは有効な算術定数または複素数式を表わしていなければならない。これに反すると，CONVERSION条件が起こる（ただし，この条件が解禁状態にある場合に限る）。定数には符号がついていてもよい。また，定数の前後にブランクがあってもよい。しかし，符号と値の間にブランクがあってはならない。また，複素数式の実数部の末尾と虚数部の直前にある符号との間にブランクがあってはならない。

ナル・ストリングの値はゼロになる。

定数は，基数，スケール，モード，精度の属性をもっているはずである。目標フィールドの属性が（たとえば，割当ての場合のように）原フィールドの属性から独立している場合には，定数は目標フィールドの属性に変換される。目標フィールドのデータとしてコード化算術データを取り上げているページで，原フィールドとして定数の属性を使用する項目を参照のこと。

ただし，中間目標フィールドが必要な場合（たとえば，演算式の値を計算する場合）は，この中間目標フィールドの属性は，ストリングの代りに精度が（15，0）の10進固定小数点整数があったときにとられるような属性となる。（したがって，コンパイラーは，ストリングに含まれている定数の属性にかかわりなく，すべての事例を処理するためのコードを生成することができる。）したがって，定数に小数部分があるとこの小数部分は失われる。目標フィールドのデータ・タイプとしてコード化算術データを取り上げているページで，原フィールドとしてFIXED DECIMALを使用する項目を参照のこと。

BIT

原フィールドのストリングは，演算式の値の計算中に変換が行なわれる場合には精度が（31，0）の無符号の2進整数であると解釈され，割当てのときに変換が行なわれる場合には精度が（56，0）の無符号の2進整数であると解釈される。割当てのときに他のものより大きい精度が許されるのは，コンパイラーが，最終的な目標フィールドを容易に判断することができるからである。目標フィールドのデータ・タイプとしてコード化算術データを取り上げているページで，原フィールドとしてFIXED BINARYを使用する項目を参照のこと。

原フィールド：　　　　目標フィールド：　コード化算術形式

原フィールドのストリングが許容精度で許される長さよりも長い場合は，左端のビットが無視される。このときに，ゼロ・ビットでないビットが失われると，結果はどうなるかわからず，この条件が解禁状態になっていれば，条件が起こる。

ナル・ストリングの値はゼロになる。

原フィールド：　　　　目標フィールド：　FIXED BINARY

FIXED BINARY

精度変換の際にも２進小数点の位置合せが行なわれるので，左端または右端でパディングや切捨てが行なわれることがある。左端のゼロ以外のビットが失われた場合，結果はどうなるかわからない。この場合，ＳＩＺＥ条件が解禁状態にあればこのＳＩＺＥ条件が起こる。

FIXED DECIMAL

原フィールドの精度が（ p_1 ，q_1 ）であるとすると，結果の精度は（ p_2 ，q_2 ）である（ただし，p_2 ＝１＋CEIL（ p_1 ＊3.32 ），q_2 ＝CEIL（ q_1 ＊3.32 ））。p_2 の計算結果が３１を越える場合は，左端の有効数字が失われることがあり，その結果ＳＩＺＥ条件が解禁状態にあればＳＩＺＥ条件が起こり，結果はどうなるかわからない。

FLOAT BINARY

この変換は，データを割り当てる場合にだけ起こり得る。精度変換は，FIXED BINARYからFIXED BINARYへの変換の場合と同じである（宣言または指示されている p_1 と，１０進小数点の位置によって示され，指数の値によって調整された q_1 が用いられる）。

FLOAT DECIMAL

この変換は，データを割り当てる場合にだけ起こり得る。精度変換はFIXED DECIMALからFIXED BINARYへの変換の場合と同じである（宣言または指示されている p_1 と，１０進小数点の位置で示され，指数の値によって調整された q_1 が用いられる）。

PICTURE（数字）
CHARACTER
BIT

目標フィールドとしてコード化算術データを取り上げているページを参照のこと。

原フィールド：　　　　目標フィールド：　FIXED DECIMAL

FIXED BINARY

原フィールドの精度が（p_1，q_1）であるとすると，結果の精度は（p_2，q_2）である（ただし，p_2 =1+CEIL（p_1／3.32），q_2 =CEIL（q_1／3.32））。

FIXED DECIMAL

精度変換の際にも10進小数点の位置合せが行なわれるので，左端または右端でパディングや切捨てが行なわれることがある。左端のゼロ以外のビットが失われたときは，結果はどうなるかわからない。この場合，SIZE条件が解禁状態にあればSIZE条件が起こる。

FLOAT BINARY

この変換は，データを割り当てる場合にだけ起こり得る。精度変換は，FIXED BINARYからFIXED DECIMALへの変換の場合と同じである（宣言または指示されたp_1と，2進小数点の位置によって示され，指数の値によって調整されたq_1が用いられる）。

FLOAT DECIMAL

この変換は，データを割り当てる場合にだけ起こり得る。精度変換は，FIXED DECIMALからFIXED DECIMALへの変換の場合と同じである（宣言または指示されたp_1と，10進小数点の位置で示され，指数の値によって調整されたq_1が用いられる）。

PICTURE（数字）
CHARACTER
BIT

目標フィールドとしてコード化算術データを取り上げているページを参照のこと。

原フィールド：　　　　　　　　　　　　　　　　目標フィールド：　FLOAT BINARY

FIXED BINARY

原フィールドの精度が（p_1，q_1）であるとすると，結果の精度はp_2である（ただし，$p_2=p_1$）。値に小数部分があれば，指数によって示される。

FIXED DECIMAL

原フィールドの精度が（p_1，q_1）であるとすると，結果の精度はp_2である（ただし，$p_2=CEIL(p_1*3.32)$）。値に小数部分があれば，指数によって示される。

FLOAT BINARY

右端にゼロをパディングすることにより結果の精度が短精度から長精度に変換されたり，右端を切り捨てることによって，結果の精度が長精度から短精度に変換されることがある。

FIOAT DECIMAL

原フィールドの精度が（p_1，q_1）であるとすると，結果の精度はp_2である（ただし，$p_2=CEIL(p_1*3.32)$）。

PICTURE（数字）
CHARACTER
BIT

目標フィールドとしてコード化算術データを取り上げているページを参照のこと。

原フィールド：　　　　目標フィールド：　FLOAT DECIMAL

FIXED BINARY

原フィールドの精度が（p_1，q_1）であるとすると，結果の精度はp_2である（ただし，p_2=CEIL（p_1／3.32)）。値に小数部分があれば指数によってこれが示される。

FIXED DECIMAL

原フィールドの精度が（p_1，q_1）であるとすると，結果の精度はp_2である（ただし，$p_2=p_1$）。値に小数部分があれば指数によってこれが示される。

FLOAT BINARY

原フィールドの精度が（p_1）であるとすると，結果の精度はp_2である（ただし，p_2=CEIL（p_1／3.32））。

FLOAT DECIMAL

右端にゼロをパディングすることにより結果の精度が短精度から長精度に変換されたり，あるいは，右端を切り捨てることにより結果の精度が長精度から短精度に変換されることがある。

PICTURE（数字）
CHARACTER
BIT

目標フィールドとしてコード化算術データを取り上げているページを参照のこと。

原フィールドのデータは，数字の形に変換されると，目標フィールドの変数のもつ既知の属性によって全面的に決められる属性をとる。PICTURE指定は，すべて，コード化算術データを暗示しており，したがって，FIXED DECIMALまたはFLOAT DECIMALを暗示している。数字を目標フィールドとする場合の以下の変換規則は，エラーのないように割当てを行なうのに必要な目標フィールドの属性を示したものである。

FIXED BINARY

原フィールドの精度が（p_1，q_1）であるとすると，目標フィールドの変数は次のような属性を暗示していなければならない。

FIXED DECIMAL （1+x+q-y，q）または
FLOAT DECIMAL （x）
ここで，　x>=CEIL（p_1/3.32），y=CEIL（q_1/3.32），q>=y

FIXED DECIMAL

原フィールドの精度が（p_1，q_1）であるとすると，目標フィールドの変数は次のような属性を暗示していなければならない。

FIXED DECIMAL （x+q-p1，q）または
FLOAT DECIMAL （x）
ここで，　x>=p，q>=q_1

FLOAT BINARY

原フィールドの精度が（p_1）であるとすると，目標フィールドの変数は次のような属性を暗示していなければならない。
FIXED DECIMAL （p，q）または
FLOAT DECIMAL （p）
ここで，p>=CEIL（p_1/3.32）であり，pおよびqの値は原フィールドの指数が表わし得る値の範囲を考慮に入れたものである。

FLOAT DECIMAL

原フィールドの精度が（p_1）であるとすると，目標フィールドの変数は次のような属性を暗示していなければならない。

FIXED DECIMAL （p，q）または
FLOAT DECIMAL （p）
ここで，p>=p_1 であり，pおよびqの値は原フィールドの指数が表わし得る値の範囲を考慮に入れたものである。

PICTURE（数字）

暗示される属性はFIXED DECIMALかFLOAT DECIMALである。このような目標フィールドを取り上げているページの該当する項目を参照のこと。

CHARACTER

目標フィールドの変数は，1文字対1文字の対応関係で（ただし，挿入文字を除く）原フィールドのストリングを収容する文字ストリングを暗示していなければならず，変換は必要でない。挿入文字は，目標フィールドへの割当てのときに取り除かれる。

原フィールド：　　　　　　　　　　目標フィールド：　PICTURE（数字）

BIT

原フィールドのストリングの長さが（n）であるものとすると，目標フィールドの変数は次のような属性を暗示していなければならない。

```
FIXED DECIMAL (1+x+q,q)または
FLOAT DECIMAL (x)
ただし,   x>=CEIL(n/3.32), q>=0
```

コード化算術形式

算術値は１０進定数に変換される。この定数は，原フィールドの属性をもとにして決められた長さをもつ中間文字ストリングに収容される。この中間文字ストリングは，CHARACTERからCHARACTERへの変換の項で示されている規則にしたがって，目標フィールドに割り当てられる。

コード化算術形式から文字ストリングへの変換規則は，リスト指示出力，データ指示出力，およびキーの値の計算（さらにREGIONALファイル）にも適用されることに注意されたい。

FIXED BINARY

２進数の精度（p_1，q_1）が，まず，１０進数の等値の精度（p，q）に変換される（ただし，p＝1＋CEIL（p_1／3.32)，q＝CEIL（q_1／3.32)）。この後は，FIXED DECIMALからCHARACTER への変換規則が適用される。

FIXED DECIMAL

原フィールドにある，精度（p，q）の１０進固定小数点数は，次のようにして変換される。

1. p＞＝q＞＝0であれば，

 - 幅がp＋3のフィールドに定数が右寄せして置かれる。
 - 先行ゼロはブランクと置き換えられる。ただし，小数の１０進小数点の直前にあるゼロは１つだけ残される。
 - 負の数の場合には，その最初の桁の直前に負符号がつけられる。値が正であれば符号はつかない。
 - 原フィールドのデータが整数である場合を除き，定数の小数部分の桁数はqである。

2. p＜qまたはq＜0であれば，定数の右にスケール因数がつく。このスケール因数は次のような形をしている。

 F｛＋｜－｝nnn，（ただし，｛＋｜－｝nnnはqの値である）。

 中間文字ストリングの長さはp＋k＋3である。ただし，kはqの値を収容するに必要な桁数である（符号やFという文字のための桁は含まない）。

算術値が複素数値の場合には，中間文字ストリングは虚数部と実数部から成っており両者はつながっている。左側が実数部であるが，この左側の部分は原フィールドの実数部と全く同じ形をとる。右側の部分は虚数部であり，この部分にはつねに符号がつきIという文字もつけられる。このような形で生成されたストリングは，複素数式であり，この式の要素の間にブランクは挿入されない。中間文字ストリングの長さは次の通りである。

p＞＝q＞＝0ならば　2＊p＋7
p＜qまたはq＜0ならば　2＊（p＋k）＋7

次に示すのは固定小数点１０進数値（実数値および複素数値）から生成される中間文字ストリングの例である。

精度　：	(5,0)	(4,1)	(4,-3)	(2,1)
値　：	2947	-121.7	-3279000	1.2+0.3I
ストリング：	'bbbb2947'	'b-121.7'	'-3279F+3'	'bbb1.2+0.3I'

FLOAT BINARY

浮動小数点２進数の精度（p_1）が，まず，等値の浮動小数点１０進数の精度（p）に変換される（ただし，p＝CEIL（p_1／3.32））。 この後は，FLOAT DECIMALからCHARACTERへの変換規則が適用される。

FLOAT DECIMAL

原フィールドにある，精度（p）の１０進浮動小数点数は，E（w，d，s）という形のE形式項目によって転送される場合と全く同様に変換される。ここで，

w（中間文字ストリングの長さ）＝p＋6
d（小数部分の桁数）＝p－1
s（有効数字の桁数）＝p

E形式項目で転送されたデータは，符号のついた２桁の指数をもつ浮動小数点１０進定数となる（「E節： 編集指示の形式項目」を参照のこと）。

算術値が複素数値の場合には，中間文字ストリングは虚数部と実数部から成っており，両者はつながっている。左側が実数部であるが，この左側の部分は原フィールドの実数部と全く同じ形をとる。右側の部分は虚数部であり，この部分にはつねに符号がつきＩという文字もつけられる。このような形で生成されたストリングは複素数式であり，この式の要素の間にブランクは挿入されない。中間文字ストリングの長さは次の通りである。

2＊p＋13

次に示すのは浮動小数点１０進数値（実数値および複素数値）から生成される中間文字ストリングの例である。

精度	:	(5)	(5)	(3)
値	:	1735＊10＊＊5	－.001663	1
ストリング	:	'b1.7350E+08'	'-1.6630E-03'	'b1.00E+00'

精度	:	(5)
値	:	17.3+1.5I
ストリング	:	'b1.7300E+01+1.5000E+00I'

PICTURE（数字）

実数の数字フィールドは，文字ストリングとして解釈され，CHARACTERからCHARACTER への変換規則に従って目標フィールドのストリングに割り当てられる。数字フィールドの中に複素数が入っている場合には，目標フィールドのストリングに割当てが行なわれる前に実数部と虚数部が連結される。

挿入文字も目標フィールドのストリングに挿入される。

CHARACTER

原フィールドのストリングは，左端から順に，目標フィールドのストリングに割り当てられる。原フィールドのストリングが目標フィールドのストリングよりも長い場合は，右端の収容しきれない文字は無視される。この際，STRINGSIZE条件が解禁状態にあればこの条件が起こる。目標フィールドのストリングが原フィールドのストリングより長い場合には，目標フィールドの右側にブランクが補充される。

BIT

ビット0が文字0になり，ビット1が文字1になる。ナル・ビット・ストリングはナル文字ストリングになる。このようにして生成された文字ストリングは，CHARACTERからCHARACTERへの変換規則に従って，目標フィールドのストリングに割り当てられる。

原フィールド：　目標フィールド：　ＢＩＴ

コード化算術形式

必要に応じて，算術値が２進数値に変換される。この際，符号や小数部分は無視される。（算術値が複素数値の場合は，虚数部も無視される。）こうして得られた２進整数は，ビット・ストリングとして取り扱かわれる。このビット・ストリングは，ＢＩＴからＢＩＴへの変換規則に従って，目標フィールドに割り当てられる。

FIXED BINARY

原フィールドの精度が（ｐ，ｑ）であるとすると，中間ビット・ストリングの長さは次の式で与えられる。

MIN(31,(p−q))

（ｐ−ｑ）が負かゼロの場合，中間ビット・ストリングはナル・ビット・ストリングである。

次に示すのは固定小数点２進数値から得られる中間ビット・ストリングの例である。

精度	：	(1)	(3)	(4,2)
値	：	1	-3	1.25
ストリング	：	'1'B	'011'B	'01'B

FIXED DECIMAL

原フィールドの精度が（ｐ，ｑ）であるとすると，中間ビット・ストリングの長さは次の式で与えられる。

MIN(31,CEIL((p−q)*3.32))

（ｐ−ｑ）が負かゼロの場合，中間ビット・ストリングはナル・ビット・ストリングである。

次に示すのは固定小数点１０進数値から得られる中間ビット・ストリングの例を示したものである。

精度	：	(1)	(2,1)
値	：	1	1.1
ストリング	：	'0001'B	'0001'B

FLOAT BINARY

原フィールドの精度が（ｐ）であるとすると，中間ビット・ストリングの長さは次の式で与えられる。

MIN(31,p)

FLOAT DECIMAL

原フィールドの精度が（ｐ）であるとすると，中間ビット・ストリングの長さは次の式で与えられる。

MIN(31,CEIL(p*3.32))

PICTURE（数字）

データは，まず，対応するPICTURE指定によって決まるスケールと精度をもつ１０進数であると解釈される。この後，このデータ項目は，FIXED DECIMALまたはFLOAT DECIMALからＢＩＴへの変換規則に従って，変換される。

原フィールド：　　　　　　　　　　　　　　　　　　　　　目標フィールド：　BIT

CHARACTER

文字0がビット0になり，文字1がビット1になる。CONVERSION条件が解禁状態にある場合，0または1以外の文字が出て来るとこの条件が起こる。ナル・ストリングはナル・ビット・ストリングになる。このようにして生成されたビット・ストリングは，原フィールドの文字ストリングと同じ長さになる。このビット・ストリングは，BITからBITへの変換規則に従って，目標フィールドに割り当てられる。

BIT

原フィールドのストリングは，左端から順に，目標フィールドのストリングに割り当てられる。原フィールドのストリングが目標フィールドのストリングより長い場合，右側の余分のビットは無視される。この際，STRINGSIZE条件が解禁状態にあればこの条件が起こる。目標フィールドのストリングが原フィールドのストリングより長い場合には，目標フィールドのストリングの右側にブランクが補充される。

算術演算のための諸表

右に掲げた諸表は，２つのオペランドをもつ算術式の値を計算すると，結果がどのような属性をもつことになるかを示したものである。式の値を計算する間に一時的な属性への変換が行なわれる場合にはその変換も示してある。結果の属性を求めたり，どのような中間変換が行なわれるかを調べたりするには，次の順序に従えばよい。

1. 図F.４ａを見る。値を計算すべき式のオペランドに対応する欄を探し出す。この欄には，１つの数字，または１つあるいは２つの文字，または，数字とその後に括弧でくくられた１つまたは２つの文字，のいずれかが入っている。数字は結果の属性を求めるのに使われ，文字は中間変換を調べるのに使われる。文字が入っていなければ，中間変換が行なわれないことを示している。文字と数字が入っている場合，中間変換を調べずに結果の属性を求めてもよいし，各中間変換を順次追って行ってもよい。
2. ａ．図F.４ａ の欄に数字だけが入っていた場合は，図F.４ｃ または図F.４ｄ を見る。どちらの図を見るかは，行なうべき演算に照して決めればよい。ともかく，このいずれかの表から，図F.４ａで求めた数に対応する属性を読み取る。これが結果の属性である。
   ｂ．図F.４ａの欄に文字だけが入っている場合は，読者の側で各中間オペランドの属性を求めないと結果の属性を出すことはできない。中間オペランドの属性を求めるには，図F.４ｂから，図F.４ａ で得られた文字に対応する記入事項（原フィールド・目標フィールド変換規則）を読み取り，ここに示されている変換規則の説明（本文）を読む。この説明から，中間オペランドの属性がわかるであろう。
   この後，中間オペランドの属性を使って，上の第１ステップと第２ステップを繰り返す。図F.４ａの欄に文字が２つ入っており，これがコンマで区切られている場合は，最初の文字は第１オペランドの中間属性に関するものであり，２番目の文字は第２オペランドの中間属性に関するものである。
   ｃ．図F.４ａの欄に数字があり，その後に括弧でくくられた文字が入っている場合は，数字を使って直接結果の属性を求めてもよいし，中間変換を順次追って行ってもよい。直接結果の属性を求めるには，ステップ２（ａ）の指示に従い，中間変換を順次追って行くにはステップ２（ｂ）の指示に従う。デバッギングの際には，後者の方法が効果があるであろうし，プログラムを書く際には前者の方法の方が便利であろう。

注： 指数演算を行なう場合には中間オペランドに関する文字を使用することはできない。指数演算の中間オペランドを求めるのにこの表を使用することはできない。

算術データの最大精度

図F.４ｃ および図F.４ｄ には精度計算式が載っている。当コンパイラーにおいては，実際の精度の値は次の最大値を越えることができない。

FIXED BINARY	３１
FIXED DECIMAL	１５
FLOAT BINARY	５３
FLOAT DECIMAL	１６

				第2オペランド							
				コード化算術形式				数字 (PICTURE)		CHARACTER (n_2)	BIT (n_2)
				FIXED		FLOAT					
				DECIMAL (p_2, q_2)	BINARY (p_2, q_2)	DECIMAL (p_2)	BINARY (p_2)	固定小数点	浮動小数点		
第1オペランド	コード化算術形式	FIXED	DECIMAL (p_1, q_1)	1	5(b)	3(c)	6(d)	w	y	w	x
			BINARY (p_1, q_1)	7(x)	2	8(d,z)	4(d)	w	y	w	x
		FLOAT	DECIMAL (p_1)	3(y)	6(d,z)	3	6(d)	w	y	w	x
			BINARY (p_1)	8(z)	4(z)	8(z)	4	w	y	w	x
	数字 (PICTURE)	固定小数点		a	a	a	a	a,w	a,y	a,w	a,x
		浮動小数点		c	c	c	c	c,w	c,y	c,w	c,x
	CHARACTER (n_1)			a	a	a	a	a,w	a,y	a,w	a,x
	BIT (n_1)			b	b	b	b	b,w	b,y	b,w	b,x

以下に示すのは，結果のモードがREALであるか，COMPLEXであるかを調べる方法である。
双方のオペランドがREALである場合，モードの変換はなく結果はREALである。
双方のオペランドがCOMPLEXである場合，モードの変換はなく結果はCOMPLEXである。
一方のオペランドがREALで他のオペランドがCOMPLEXである場合，REALのオペランドがCOMPLEXに変換され，結果はCOMPLEXである。ただし，これには次の２つの例外がある。例外は指数演算に関するもので，第２オペランド（指数）がFIXED（p，0）変数または固定小数点１０進整数定数の場合である。この場合，式の値を計算する前にモードの変換が行なわれることはないが，結果はCOMPLEXである。

図F.4a　算術演算のための基本表

第1オペランドのコード	第2オペランドのコード	目標フィールド
a	w	FIXED DECIMAL
b	x	FIXED BINARY
c	y	FLOAT DECIMAL
d	z	FLOAT BINARY

図F.4b　参照すべき変換規則

コード	結果の属性	結果の精度		
		加算または減算	乗算	除算
1	FIXED DECIMAL(p,q)	$p=1+MAX(p_1-q_1, p_2-q_2)+q$ $q=MAX(q_1, q_2)$	$p=p_1+p_2+1$ $q=q_1+q_2$	$p=15$ $q=15-((p_1-q_1)+q_2)$
2	FIXED BINARY(p,q)			$p=31$ $q=31-((p_1-q_1)+q_2)$
3	FLOAT DECIMAL(p)	$p=MAX(p_1, p_2)$		
4	FLOAT BINARY(p)			
5	FIXED BINARY(p,q)	$p=1+MAX(r-s, p_2-q_2)+q$ $q=MAX(s, q_2)$	$p=1+r+p_2$ $q=s+q_2$	$p=31$ $p=31-((r-s)+q_2)$
		ここで, $r=1+p_1*3.32$, $s=q_1*3.32$		
6	FLOAT BINARY(p)	$p=MAX(p_1*3.32, p_2)$		
7	FIXED BINARY(p,q)	$p=1+MAX(p_1-q_1, r-s)+q$ $q=MAX(q_1, s)$	$p=1+p_1+r$ $q=q_1+s$	$p=31$ $p=31-((p_1-q_1)+s)$
		ここで, $r=1+p_2*3.32$, $s=q_2*3.32$		
8	FLOAT BINARY(p)	$p=MAX(p_1, p_2*3.32)$		

図F. 4c　加算，減算，乗算，除算の結果の表

コード		結果の属性
1	3	FLOAT DECIMAL(p) ［ただし，特殊ケースAまたはCに該当しない場合］ $p=MAX(p_1, p_2)$
2	4	FLOAT BINARY(p) ［ただし，特殊ケースBまたはCに該当しない場合］ $p=MAX(p_1, q_2)$
5	6	FLOAT BINARY(p) ［ただし，特殊ケースAまたはCに該当しない場合］ $p=MAX(p_1*3.32, p_2)$
7	8	FLOAT BINARY(p) ［ただし，特殊ケースBまたはCに該当しない場合］ $p=MAX(p_1, (p_2*3.32))$

特殊ケース(下の注を参照のこと)			
ケース	第1オペランド	第2オペランド	結果の属性
A	FIXED DECIMAL (p_1, q_1)	無符号の整数定数 (値＝n)	FIXED DECIMAL(p,q) ［ただし，p<＝15］ $p=(p_1+1)*n-1$ $q=q_1*n$
B	FIXED BINARY (p_1, q_1)	無符号の整数定数 (値＝n)	FIXED BINARY(p,q) ［ただし，p<＝31］ $p=(p_1+1)*n-1$ $q=p_1*n$
C	FLOAT (p_1)	FIXED $(p_2, 0)$	FLOAT (p_1) (基数は第1オペランドの基数)

次に示すように，さらにx**yの特殊ケースがある。

REALモード:

x＝0，y＞0ならば，結果はゼロ

x＝0，y<＝0ならば，ERROR条件が起こる'

x<0，yがFIXED(p,0)でないならば，ERROR条件が起こる。

COMPLEXモード:

x＝0，yの実数部＞0，yの虚数部＝0ならば，結果はゼロ

x＝0，yの実数部<＝0またはyの虚数部¬＝0ならば，ERROR条件が起こる。

x¬＝0，yの実数部＝yの虚数部＝0ならば，結果は1

図F. 4d　指数演算の結果の表

比較演算のための諸表

右に掲げた諸表は，比較演算の２つのオペランドが比較の前にどのような属性に変換されるかを示したものである。表には，行なうべき比較のタイプも示されている。
まず，図F．５ａ を見る。値を計算すべき式の２つのオペランドに対応する欄をこの表の中から見つけ出す。この欄の中にはコンマで区切られた２つの数字が入っているはずである。この数字が図F．５ｂを見る際の鍵になっている。図F．５ｂの右側の欄には各々のオペランドが変換される属性が示されている。図F．５ａ の各々の欄の最初の数字は，第１オペランドがどのような属性に変換されるかを調べる鍵になっており，２番目の数字は，第２オペランドがどのような属性に変換されるかを調べる鍵になっている。たとえば，変数が次に示すように宣言されている比較について考えて見よう。

```
DECLARE ITEM CHARACTER(5),
        STANDARD FIXED BINARY (15,0);
                 .
                 .
                 .
IF ITEM¬=STANDARD THEN DO;
```

図F．５ａ を見ると，第１オペランドの属性がCHARACTERで第２オペランドの属性がFIXED BINARYである欄には，１３，１と記されている。図F．５ｂの１３という所を見ると，FIXED BINARY（３１，０）と記されている。したがって，ITEMはこのような属性をもつコード化算術形式に変換されることになる。図F．５ｂの１という所を見ると「変換は行なわれない」と記されている。したがって，STANDARDは変換されないわけである。図F．５ｂの１３の所も１の所も代数的な比較が行なわれると指示されている（図F．５ｂにおいて２つのオペランドに対応する数字は，つねに同じ比較のタイプを示すはずである）。以上見て来たように，ITEMは，FIXED BINARY（３１，０）に変換され，次いで，属性が初めからFIXED BINARY（１５，０）であるSTANDARDと代数的に比較される。

算術データの最大精度

図F．５ｂには，精度計算式が示されている。当コンパイラーのもとでは，実際の精度の値は次に示す最大値を越えることができない。

FIXED BINARY	３１
FIXED DECIMAL	１５
FLOAT BINARY	５３
FLOAT DECIMAL	１６

				第2オペランド							
				コード化算術形式				数字 (PICTURE)		CHAR-ACTER (n_2)	BIT (n_2)
				FIXED		FLOAT					
				DECIMAL (p_2, q_2)	BINARY (p_2, q_2)	DECIMAL (p_2)	BINARY (p_2)	固定小数点	浮動小数点		
第1オペランド	コード化算術形式	FIXED	DECIMAL (p_1, q_1)	1,1	4,1	5,1	8,1	1,10	5,11	1,12	4,13
			BINARY (p_1, q_1)	1,4	1,1	7,6	7,1	1,4	7,6	1,13	1,13
		FLOAT	DECIMAL (p_1)	1,5	6,7	1,1	6,1	1,5	1,11	1,14	6,9
			BINARY (p_1)	1,8	1,7	1,6	1,1	1,8	1,6	1,15	1,9
	数字 (PICTURE)	固定小数点		10,1	4,1	5,1	8,1	10,10	5,11	10,12	4,13
		浮動小数点		11,5	6,7	11,1	6,1	11,5	11,11	11,14	6,9
	CHARACTER (n_1)			12,1	13,1	14,1	15,1	12,10	14,11	2,2	2,2
	BIT (n_1)			13,4	13,1	9,6	9,1	13,4	9,6	9,2	3,3

一方のオペランドがCOMPLEXで他方のオペランドがREALである場合，比較の前にREALのオペランドがCOMPLEXに変換される。

図F.5a　比較演算のための基本表

コード	比較のタイプ	比較の目標フィールドの属性
1	代 数	変換は行なわれない
2	文 字	CHARACTER (MAX(n_1, n_2)), ただし, (n_1),(n_2) はそれぞれ第1, 第2オペランドの長さ
3	ビット	BIT (MAX(n_1, n_2)), ただし, (n_1),(n_2)はそれぞれ第1 ,第2オペランドの長さ
4	代 数	FIXED BINARY (1+p*3.32, q*3.32), ただし, (p,q)は変換されるオペランドの精度 (オペランドが数字 (PICTURE)の形である場合には, 注1を見よ)
5	代 数	FLOAT DECIMAL(p), ただし, (p,q)は変換されるオペランドの精度 (オペランドが数字 (PICTURE) の形である場合には, 注1を見よ)
6	代 数	FLOAT BINARY (p*3.32), ただし, (p)は変換されるオペランドの精度 (オペランドが数字 (PICTURE) の形である場合には, 注1を見よ)
7	代 数	FLOAT BINARY (p), ただし, (p,q)は変換されるオペランドの精度 (オペランドが数字 (PICTURE) の場合には, 注1を見よ)
8	代 数	FLOAT BINARY (p*3.32), ただし, (p,q)は変換されるオペランドの精度 (オペランドが数字 (PICTURE) の形である場合には, 注1を見よ)
9	代 数	FLOAT BINARY(31)
10	代 数	FIXED DECIMAL (精度は, 変換されるオペランドのPICTURE指定で暗示されている精度)
11	代 数	FLOAT DECIMAL (精度は, 変換されるオペランドのPICTURE指定で暗示されている精度)
12	代 数	FIXED DECIMAL (15,0)
13	代 数	FIXED BINARY (31,0)
14	代 数	FLOAT DECIMAL (15)
15	代 数	FLOAT BINARY (50)
16	代 数	FLOAT DECIMAL (10)

注1： 変換されるオペランドが数字の形をとっている場合には, その精度はPICTURE指定で暗示されている精度である。

図F.5b 比較演算のタイプと目標フィールド

G節：　組込み関数と擬似変数

この節には，プログラマーが使用することのできる組込み関数がすべての収録されており，アルファベット順に並べてある。擬似変数として使用することのできる組込み関数に関しては，その擬似変数の機能について説明する項を設けてある。

組込み関数参照の一般的な形式は，次に示す通りである。

関数名 〔（x|x_1，x_2，・・，x_n）〕

ここで，xまたはx_1，x_2，・・・，x_nは，所要の引数を表わしている。ある種の関数においては，必要に応じて指定しても指定しなくてもよい引数が1つ以上ある。たとえば，

SUBSTR（x_1，x_2〔，x_3〕）

以下アルファベット順に各々の関数について説明するわけであるが，各々の関数の見出しとしてその関数参照の一般的な形式を掲げてある（擬似変数参照はつねにその関数参照と同じ形である）。一般に，各関数の説明では次のような点に触れてある。

1. 返される値についての説明。
2. 引数についての詳しい説明。
3. 関数使用上の他の制約。
4. 該当する関数に関しては，擬似変数としての機能の説明。

組込み関数の分類

組込み関数を分類するにあたっては，組込み関数の使用目的がPL／Iのどの機能に関係しているかという観点から分類することができる。この分類方法にしたがって，組込み関数を次のように分類することができる。

ストリング取扱い組込み関数
算術組込み関数
数学組込み関数
配列取扱い組込み関数
条件取扱い組込み関数
記憶域制御組込み関数
非同期入出力組込み関数
STREAM入出力組込み関数
日付・時間機能組込み関数

最初の4つが計算関係の組込み関数である。

ストリング取扱い組込み関数

これらの関数は，ビット・ストリングおよび文字ストリングの処理を簡易化するためのものであり，次のようなものがある。

BIT	REPEAT
BOOL	STRING
CHAR	SUBSTR
HIGH	TRANSLATE
INDEX	UNSPEC
LENGTH	VERIFY
LOW	

算術組込み関数

これらの関数は，プログラマーの側で，基数，スケール，モード，および精度の変換を直接的に，また基本的な算術演算を行なっている間に，制御することができるようにするためのものである。このほかに，算術値の個々の性質を調べるための組込み関数もある。たとえば，SIGN関数は算術値の符号を調べるのに使用される。このような組込み関数としては次のようなものがある。

ABS	IMAG
ADD	MAX
BINARY	MIN
CEIL	MOD
COMPLEX	MULTIPLY
CONJG	PRECISION
DECIMAL	REAL
DIVIDE	ROUND
FIXED	SIGN
FLOAT	TRUNC
FLOOR	

数学組込み関数

これらの関数は，標準的な算術演算を行なうためのものであり，次のようなものがある。

ACOS	LOG
ASIN	LOG2
ATAN	LOG10
ATAND	SIN
ATANH	SIND
COS	SINH
COSD	SQRT
COSH	TAN
ERF	TAND
ERFC	TANH
EXP	

配列取扱い組込み関数

これらの関数はすべて，配列引数を対象としており，ある配列の1つの特性値を返すものである。これらの関数には，次のものがある。

ALL	LBOUND
ANY	POLY
DIM	PROD
HBOUND	SUM

条件取扱い組込み関数

これらの関数は，プログラマーの側で，解禁状態にある例外条件に起因する割込みについて調べることができるようにするためのものである。各々の関数は，当該関数が対象としている特定の例外条件が起ったときだけ入ることのできるオン・ユニットの範囲内，あるいは，システムの標準処置としてERRORないしFINISH 条件が起った場合にはこれらの条件のためのオン・ユニットの範囲内でだけ，定義されている値を返す。このような組込み関数は，次の通りである。

DATAFIELD	ONFILE
ONCHAR	ONKEY
ONCODE	ONLOC
ONCOUNT	ONSOURCE

記憶域制御組込み関数

これらの関数は，基底付変数または被制御変数に関する特殊な値を返したり，基底付変数の所在場所を探したりするためのものである。このような関数は，次の通りである。

ADDR	NULL
ALLOCATION	OFFSET
EMPTY	POINTER

非同期入出力組込み関数

これらの関数は，プログラマーの側で，事象変数の現在の状態について調べることができるようにするためのものであり，次のようなものがある。

COMPLETION
STATUS

STREAM入出力組込み関数

これらの関数は，プログラマーの側で，ファイルの現在の状態を調べることができるようにするためのものであり，次のようなものがある。

COUNT
LINENO

引数の変換

引数の変換が行なわれる可能性があるのは，計算関係の組込み関数の場合だけである。これらの組込み関数に渡す引数は，演算式であってもよい。引数として渡される式（この中に組込み関数への参照があってもよい）は，まずその値が計算され，データ変換規則に従って当該組込み関数に適した形に変換される。各々の引数が要求するデータ・タイプについては，個々の関数の説明の項で触れる。

ストリング取扱い組込み関数

この種の関数の中には，ストリング引数ばかりでなく算術引数も必要とするものもある。算術引数は，ストリングの長さを表わすものであるから，整数であるか整数に変換可能なものでなければならない。ストリング引数を算術式で表わすこともできる。この場合，この算術式は，データ変換規則または関数の説明の項で示されている規則に従って，ストリングに変換される。プログラマーは，これらの変換の結果，所望のストリング・タイプで関数が操作されるよう確認しなければならない。

算術組込み関数

これらの関数の中には，結果のデータ・タイプを1つ以上の引数をもとにして決定するものもある。引数のデータ・タイプが違っているときは，データ変換規則に従って，一方の引数が他の引数に合致するよう変換されるので結果は同じデータ・タイプになる。結果のデータ属性が引数のデータ属性と一致しない場合（たとえば，FLOOR 組込み関数）については，個々の関数の説明の項でその規則について説明する。

Nという文字は，固定小数点数の結果に関する許容最大精度を表わすのに用いられている。Nの値は下に示す通りである。

FIXED DECIMAL	15
FLOAT DECIMAL	16
FIXED BINARY	31
FLOAT BINARY	53

数学組込み関数

これらの関数はすべて浮動小数点数値を対象としており，浮動小数点数の結果を返す。したがって，浮動小数点数でない引数があると，この引数は浮動小数点数に変換される。

配列取扱い組込み関数

これらの関数で必要とされる引数の変換については，各関数の説明の項で触れる。

数学関数の正確度

結果の正確度は，次の２つの要因に左右される。

1. 引数の正確度

2. 算法の性能

ほとんどの引数には誤差が含まれている。さらに，関数の値を計算する前に，ある引数の誤差が何段階かにわたって積み重ねられていることもある。入力変換を経たばかりの新しいデータにも小さな誤差が含まれていることがある。引数の誤差が結果の正確度に及ぼす影響は，ひとえに数学関数の性格によるものであって，結果を計算するための算法とは無関係である。本書では，この種類の誤差についてはこれ以上触れないことにする。

各数学関数の性能の統計が，図G.1（3ページにわたって収録）に示されている。この値は，引数には誤差が含まれていないという前提に立ったものである。

各関数ごとに，有効な引数の範囲またはその範囲の代表的な部分についての正確度の値が与えられている。どの関数の場合も，表に示されている数字は，検討の対象とされている関数および範囲にとって最も意味のあるものである。

たとえば，一組の結果をもとにした相対誤差の自乗の平均値の平方根（RMS）と相対誤差の最大値は，一般に役に立つ意味のある数字である。しかし，これらの数字は，関数の値がゼロになるような範囲では役に立たない。と言うのも，この場合，引数の値に極めて小さな誤差が含まれていても，結果の相対的な大きさに無限の変動をもたらすことがあるからである。たとえば，SINx においてxの値がπに近づいたときのような状態になる。このような範囲においては，絶対誤差を問題にする方が妥当である。

図G.1で示されている値は，関数の性格に応じて各範囲において一様分布または指数分布をなしている5000　の引数から得られた無作為の結果をもとにしたものである。誤差の最大値として挙げられている値は，各々ある特定のサンプルに関するものであり，誤差の真の最大値がどの程度のものかを知るための指針にすぎないことに注意されたい。

短精度および長精度の浮動小数点数の各々の結果について，誤差の最大値およびRMS値を与えてある。

関数の各々の範囲について，相対誤差または（必要に応じて）絶対誤差の最大値とRMS値を与えてある。これらの値は次のようにして定義されたものである。

f（x）　＝　関数の真の値

g（x）　＝　関数の計算された値

であるものとすれば，結果の絶対誤差は，

ABS(f(x)－g(x))

となり，結果の相対誤差は，

ABS(f(x)－g(x))／f(x)

となる。
ここで，結果のサンプル数がn であるものとすると，絶対誤差のRMS値は次の式で与えられる。

$$\mathrm{SQRT}(\Sigma_{i=1}^{n}((f(x)-g(x))**2)/n)$$

相対誤差のRMS値は次の式で与えられる。

$$\mathrm{SQRT}(\Sigma_{i=1}^{n}(((f(x)-g(x))/f(x))**2)/n)$$

引数としての配列および構造

引数として構造を受け入れることのできる関数はADDR, ALLOCATION，およびSTRING だけである。

引数をもつことのできる組込み関数はすべて，引数として配列を受け入れることができる。ADDR，ALLOCATION，STRING，および配列取扱い関数は，１つの値を返すだけであるが，これら以外の関数はすべて値の配列を返す。したがって，たとえばSUBSTRのような関数を使用する場合は，どの引数も配列であってよい（２つ以上の引数が配列である場合は，これらの配列の限界が同じでなければならない）。この機能は，制御変数によって変更される添字付配列参照を引数として１つ以上もっているDOループの中に関数参照を置くのと同じである。

ナル引数

引数を必要としない組込み関数も多い（主として,条件取扱い関数)。このような関数については，BUILTIN属性で明示的にこれを宣言するか，関数参照の中にナル引数リストを含めて（たとえば，ONCHAR( ) 文脈の上からこれを宣言しなければならないと言う規則があるのに注意されたい。このいずれかの措置をとらないと，コンパイラーはその名前を組込み関数名として認定することができない。

擬似変数

受取り側のフィールドを表わすのに使用することができる組込み関数もある。組込み関数をこのような形で使用した場合，これは擬似変数である。擬似変数は，割当てステートメントまたはDOステートメントの等号の左側に記入することができる。また，GETステートメントのデータ・リストに擬似変数を記入することもできるし，KEYTO, STRING, またはREPLYオプションのストリング名として擬似変数を使用することもできる。

擬似変数はすべて組込み関数でもあるから，擬似変数に関しては該当する組込み関数の説明の項で簡単に触れるにとどめる。

擬似変数は，ネストすることができない点に注意されたい。たとえば，次のステートメントは無効である。

```
UNSPEC(SUBSTR(A,1,2))='00'B;
```

以下に示すのは擬似変数である。

COMPLETION	REAL
COMPLEX	STATUS
IMAG	STRING
ONCHAR	SUBSTR
ONSOURCE	UNSPEC

関　数　名	引数のモード	範　　囲	短精度浮動小数点数 相対誤差 *10**8		長精度浮動小数点数 相対誤差 *10**17	
			RMS	最大	RMS	最大
ACOS(x)	実　数					
ASIN(x)	実　数					
ATAN(x)	実　数	ABS(x)<1	13	90	4	21
		全　域[2]	25	99	5	21
	複素数	全　域[2]	21	110	5	44
ATAN(x_1, x_2)	実　数	ABS(x_1)≦1, ABS(x_2)≦1	29	160	7	36
ATAND(a)	(実数引数のATAN と同じ)					
ATANH(x)	実　数	ABS(x)≦0.2	46	110	−	−
		ABS(x)≦0.9	39	120	9	−
		ABS(x)≦0.25	−	−	6	21
		ABS(x)≦0.95	−	−	9	25
	複素数	全　域[2]	22	120	6	41
COS(x)	実　数[1]	0≦x≦π	4	12	7	27
		-10≦x<0, π<x≦10	4	12	7	27
		10<ABS(x)≦100	4	11	100	270
	複素数[3]	ABS(a)≦10, ABS(b)≦1	120	320	31	380
COSD(x)	(実数引数のCOSと同じ)					
COSH(x)	実　数	ABS(x)≦1	41	96	−	−
		1<ABS(x)<2	25	72	−	−
		ABS(x)≦170	20	82	−	−
		ABS(x)≦17	−	−	11	39
		ABS(x)≦5	−	−	11	38
	複素数[3]	ABS(a)≦10, ABS(b)≦1	97	310	25	73

[1] このRMS値と最大値は絶対誤差に関するものである。

[2] 指数分布。これ以外はすべて一様分布である。

[3] ここで，a+i*b=x

[4] ここに示した各々の数字は関数の特異点に近いある特定の点に対応するものであり，このような範囲においては誤差を調整することはできない。

図 G. 1.　数学組込み関数の性能の統計値(3の1)

関数名	引数のモード	範囲	短精度浮動小数点数 相対誤差 *10**8		長精度浮動小数点数 相対誤差 *10**17	
			RMS	最大	RMS	最大
ERF(x)	実数	ABS(x)≦1	12	85	3	19
		1<ABS(x)<2.04	4	11	1	3
		2.04<ABS(x)<3.9192	3	6	—	—
		2.04<ABS(x)<6.092	—	—	1	1
ERFC(x)	実数	-6<x<0	30	94	7	21
		0<x<1	13	69	3	15
		1<x≦2.04	37	200	9	43
		2.04<x<4	37	130	9	33
		4≦x<13.3	820	1500	200	350
EXP(x)	実数	-1<x<1	13	44	5	21
		全域	12	46	5	43
	複素数	ABS(a)≦170 ABS(b)≦π/2	65	240	—	—
		ABS(a)≦170 π/2<ABS(b)≦20	63	230	—	—
		ABS(a)<1 ABS(b)<π/2	—	—	19	61
		ABS(a)<20 ABS(b)<20	—	—	20	82
LOG(x)	実数	0.5<x<2.0[2] を除く	12	84	5	34
		0.5<x<2.0[1]	3	7	2	5
	複素数	全域[2]	40	190	13	54
LOG2(x)	実数	0.5<x<2.0[2] を除く	34	98	9	43
		0.5<x<2.0[1]	23	48	3	6
LOG10(x)	実数	0.5<x<2.0[2] を除く	22	110	7	32
		0.5<x<2.0[1]	2	7	1	3

(注については，299頁を参照のこと。)

図 G. 1. 数学組込み関数の性能の統計値(3の2)

関数名	引数のモード	範囲	短精度浮動小数点数 相対誤差 *10**8		長精度浮動小数点数 相対誤差 *10**17	
			RMS	最大	RMS	最大
SIN(x)	実数[1]	ABS(x)≦π/2	5	12	2	8
		π/2<ABS(x)≦10	4	13	32	240
		10<ABS(x)≦10	4	12	93	270
	複素数[3]	ABS(a)≦10,ABS(b)≦1	120	340	200	11300
SIND(x)	(実数引数のSINと同じ)					
SINH(x)	実数	ABS(x)≦1	20	88	–	–
		1<ABS(x)<2	25	100	–	–
		ABS(x)≦170	20	82	–	–
		ABS(x)≦17	–	–	10	36
		ABS(x)<0.881374	–	–	4	20
		0.881374<ABS(x)≦5			10	35
	複素数	ABS(a)≦10,ABS(b)≦1	82	280	23	64
SQRT(x)	実数	全域[2]	13	48	3	11
	複素数	全域[2]	54	220	13	49
TAN(x)	実数[4]	ABS(x)≦π/4	29	160	65	570
		π/4<ABS(x)<π/2	37	150	–	–
		π/4<ABS(x)<1.5	–	–	47	230
		π/2<ABS(x)≦10	32	480	–	–
		1.5<ABS(x)≦10	–	–	8400	473000
		10<ABS(x)≦100	31	140	7900	271000
	複素数[3]	ABS(a)<1,ABS(b)<9	43	240	15	62
TAND(x)	(実数引数のTANと同じ)					
TANH(x)	実数	ABS(x)≦0.7	15	78	–	–
		0.7<ABS(x)≦9.011	4	29	–	–
		ABS(x)≦0.54931	–	–	4	19
		0.54931<ABS(x)≦20.10[1]	–	–	1	16
	複素数[3]	ABS(a)<9,ABS(b)<1	43	270	15	63
(注については,299頁を参照のこと。)						

図 G. 1. 数学組込み関数の性能の統計値(3の3)

ABS(x) 算術

ABSは，与えられた式xの絶対値を返す。xが実数の場合，ABSが返す値はxの正の値である。xが複素数の場合，ABSが返す値は実数部および虚数部各々の自乗の和の正の平方根である。

xが精度（p，q）の固定小数点複素数の場合，結果の精度は次の式で与えられる。

(MIN(N，p+1)，q)

ここで，Nは許容される最高の桁数である。

ACOS(x) 算術

ACOSは，与えられた値xのarc COS（COS^{-1} x）の値を表わすラジアン単位の浮動小数点数値を返す。

xは実数でなければならず，しかもxの絶対値は1以下でなければならない。すなわち，ABS(x)<=1。返される結果は次の範囲の値である。

0<=ACOS（x）<=π

ADD(x_1，x_2，x_3〔，x_4〕) 算術

ADDは，値x_1とx_2の和をx_3およびx_4で指定されている精度で返す。

x_1とx_2は加え合わせるべき値である。

x_3 加算を行なう際に確保すべき桁数を指定するための無符号の10進整数定数。この値は当コンパイラーのもとでの許容最大精度を越えてはならない。

x_4 結果のスケール因数を指定するための10進整数定数（必要に応じて符号がつけられる）。結果が固定小数点数の場合は，x_3を与えたらx_4も与えなければならない。結果が浮動小数点数の場合は，x_3しか与えることができない。

ADDR(x) 記憶域制御

ADDRは，与えられた変数xが割り振られている記憶場所を識別するためのポインター値を返す。

x 任意のデータ・タイプおよび任意の編成をもち，任意の記憶域クラスに割り振られている変数。ただし，次のような変数を除く。

1. UNALIGNED 属性をもつ固定長ビット・ストリングであるBASED変数，DEFINED変数，パラメーター変数，添字付変数，または構造基本要素変数。

2. UNALIGNED 属性をもつ固定長ビット・ストリングを最初の基本要素としている小構造（ただし，この最初の要素が，同時に，この小構造の属している大構造の最初の要素でもある場合を除く）。

3. DEFINED属性をもつ大構造またはパラメーターである大構造のうち，その最初の要素がUNALIGNED 属性の固定長ビット・ストリングであるもの。

4. 連続している記憶域に収容されていない変数。

xがデータの集合体である場合には，返される値はその最初の要素を指している。

xが長さの変わるストリングである場合には，返される値はその接頭語（2バイト）を指している。

xが記憶域である場合には，返される値はその制御情報を指している。

xがまだ割り振られていない被制御変数である場合には，ナル・ポインター値が返される。

xがパラメーターである場合，見かけ引数が作成されているならば，返される値はこの見かけ引数を指している。上の4番目の条件から，xがパラメーターであるならばこのパラメーターはCONNECTED属性をもつものでなければならないという点に注意されたい。

ALL(x) 配列取扱い

ALLは，与えられた配列xのすべての要素の中に，あるビットが存在しており，しかもそのビットが1である場合にそれと対応する各ビットが1になっているビット・ストリングを返す。結果の長さは一番長い要素の長さに等しい。

xがビット・ストリング配列でない場合，xはビット・ストリングに変換される。xがiSUBで定義されたものであってはならない。

ALLOCATION(x) 記憶域制御

ALLOCATIONは，与えられた被制御変数xの世代数を表わす省略時精度の固定小数点2進整数を返す。

x 被制御変数の名前。この名前は，レベル1の添字のついていない名前でなければならない。

xが割り振られていない場合，結果はゼロである。

ANY(x)　　配列取扱い

ANYは，与えられた配列xのいずれかの要素に，あるビットが存在しており，しかもこのビットが1である場合にこれと対応するビットが1であるようなビット・ストリングを返す。結果の長さは一番長い要素の長さに等しい。

xがビット・ストリング配列でない場合，xはビット・ストリングに変換される。xはiSUBで定義されたものであってはならない。

ASIN(x)　　数学

ASINは，与えられた値xのarc sin($sin^{-1}x$)の値を表わすラジアン単位の浮動小数点数値を返す。

xは実数でなければならず，またその絶対値は1以下でなければならない。すなわち，ABS(x)<=1。結果は次の範囲の数値である。

−π／2<=ASIN(x)<=π／2

ATAN(x_1〔，x_2〕)　　数学

ATANは，与えられた値x_1または与えられた比x_1/x_2のarc tan($tan^{-1}x_1$または$tan^{-1}x_1/x_2$)の値を表わすラジアン単位の浮動小数点数値を返す。

x_1だけが指定されており，しかもそれが実数である場合の結果は，次の範囲の数値である。

−π／2<ATAN(x_1)<π／2

x_1だけが指定されており，それが複素数である場合，x_1は＋iまたは−iであってはならない。この場合，結果は次の式で与えられる。

−i＊ATANH(i＊x_1)

x_1とx_2が指定されている場合には，両者は共に実数でなければならない。x_1とx_2が共にゼロのときは，誤りである。x_1とx_2がこれ以外の任意の値をとる場合，結果は以下の式で与えられる。

x_2＞0ならば　arctan(x_1／x_2)

x_2＝0，x_1＞0ならば　π／2

x_2＝0，x_1＜0ならば　−π／2

x_2＜0，x_1≧0ならば　π＋arctan　(x_1／x_2)

x_2＜0，x_1＜0ならば　−π＋arctan(x_1／x_2)

ATAND(x_1〔，x_2〕)　　数学

ATANDは，与えられた値x_1または与えられた比x_1/x_2のarc tan($tan^{-1}x_1$または$tan^{-1}x_1/x_2$)の値を表わす度単位の浮動小数点数値を返す。

x_1だけが指定されている場合，これは実数でなければならない。結果は次の範囲の値である。

−90<ATAND(x_1)<90

x_1とx_2が指定されている場合には，両者は共に実数でなければならない。結果は，関数ATANを用いた次の式で与えられる。

180／π＊ATAN(x_1，x_2)

ATANH(x)　　数学

ATANHは，与えられた値xのarc tanh($tanh^{-1}x$)の値を表わす浮動小数点数値を返す。

xが実数である場合，その絶対値は1より小さくなければならない。すなわち，ABS(x)<1。

xが複素数である場合，xは＋1または−1であってはならない。この場合，結果は次の式で与えられる。

LOG((1＋x)／(1−x))／2

BINARY(x_1〔，x_2〔，x_3〕〕)　　算術

BINARYは，x_2およびx_3によって指定されている精度をもつ，与えられた値x_1の2進数表現を返す。

x_1　2を基数とする表現に変換すべき値。

x_2　変換操作の間維持すべき桁数を指定するための無符号の10進整数。x_2は当コンパイラーのもとでの許容最大精度を越えていてはならない。

x_3　結果のスケール因数を指定するための10進整数(符号をつけてもつけなくてもよい)。結果が固定小数点数であり，しかもx_2を与える場合には，x_3も与えなければならない。結果が浮動小数点数である場合には，x_2しか与えることができない。x_2とx_3を共に省略した場合は，結果の精度は，基数変換規則に準じて決められる。

BIT(x_1〔,x_2〕)　　ストリング取扱い

BITは，与えられた値x_1のビット・ストリング表現を返す。

x_1　変換すべき式。

x_2　結果として得られるビット・ストリングの長さを指定するための整数に変換することのできる式。必要に応じて，x_2は精度（15，0）の2進整数に変換される。x_2を省略した場合，ビット・ストリングの長さは，タイプ変換規則に従って決定される。

BOOL(x_1，x_2，x_3)　　ストリング取扱い

BOOLは，x_3によって指定されているブール演算をビット・ストリングx_1およびx_2に行なった結果を表わすビット・ストリングを返す。結果の長さは，オペランドx_1とx_2のうち長い方のオペランドの長さになる。

x_1とx_2は，ビット・ストリング式，または，ビット・ストリングに変換することのできる式である。

x_3は，4ビットからなるビット・ストリングである。次に示すように，この各々のビットで，x_1のあるビットとx_2のこれに対応するビットとを比較した結果を指定する。

x_1	x_2	結果
0	0	x_3のビット1
0	1	x_3のビット2
1	0	x_3のビット3
1	1	x_3のビット4

x_1の長さとx_2の長さが異なる場合，短い方のビット・ストリングの右側にゼロが補なわれて，長い方のビット・ストリングの長さに揃えられる。x_3が長さ4のビット・ストリング式でない場合，x_3はビット・ストリングに変換され，右側でゼロの補充または切捨てが行なわれる。

CEIL(x)　　算術

CEILは，与えられた値x以上の最小の整数を返す。xは実数でなければならない。

xが精度（p，q）の固定小数点数であるとすると，結果の精度は次の式で与えられる。

$$(MIN(N,MAX(p-q+1,1)),0)$$

ここで，Nは許容される最大の桁数である。

CHAR(x_1〔,x_2〕)　　ストリング取扱い

CHARは，与えられた値x_1の文字ストリング表現を返す。

x_1　変換すべき式。

x_2　結果として得られる文字ストリングの長さを指定するための整数に変換することのできる式。必要に応じて，x_2は精度（15,0）の2進整数に変換される。x_2を省略した場合，文字ストリングの長さは，タイプ変換規則に従って決定される。

COMPLETION(x)　　非同期入出力

COMPLETIONは，与えられた事象xの完了値を表わす1つのビットを返す。与えられた事象が未完了の場合には，'0'Bが返される。与えられた事象が完了している場合には，'1'Bが返される。

COMPLETION擬似変数

この擬似変数は，与えられた事象xの完了値をセットする。xは非活用状態になければならない。この擬似変数への割当ての間に割込みが起こることはない。

COMPLEX(x_1，x_2)　　算術

省略形:　CPLX(x_1，x_2)

COMPLEXは，与えられた2つの値x_1およびx_2から作られた複素数値を返す。

x_1　結果の実数部となる実数値。

x_2　結果の虚数部となる実数値。

x_1とx_2の基数またはスケールが異なる場合は，データ変換規則に従って，一方の基数またはスケールが他方の基数またはスケールに合致するよう変換される。したがって，その結果の実数部と虚数部の基数ならびにスケールは同じになる。x_1とx_2は共に実数でなければならない。

結果が固定小数点数の場合，その精度は次の式で与えられる。

$$(MIN(N,MAX(p_1-q_1,p_2-q_2)+MAX(q_1,q_2)),MAX(q_1,q_2))$$

ここで，(p_1，q_1)，(p_2，q_2)はそれぞれx_1とx_2の精度であり，Nは許容される最大の桁数である。

必要な変換を行なった後の引数が浮動小数点数であり，その精度が p_1 ，p_2 であるとすると，結果の精度はMAX(p_1 ，p_2)になる。

COMPLEX擬似変数

この擬似変数は，ある複素数値の実数部を変数 x_1 に割り当て，虚数部を変数 x_2 に割り当てる。この擬似変数には，複素数値しか割り当てることができない。x_1 が配列の場合，x_2 は要素でも配列でもよい。x_1 および x_2 が配列の場合，双方の配列の限界が等しくなければならない。

CONJG(x)　　算術

CONJGは，与えられた複素数値xの共役複素数値を返す。ある複素数値の共役複素数値とは，もとの複素数の虚数部の符号を逆にして得られる値である。xが実数の場合，xは複素数に変換される。

COS(x)　　数学

COSは，与えられた値xのcosの値(cos x)を表わす浮動小数点数値を返す。

x　　ラジアンを単位とする値をもつ式。

xが複素数の場合，結果は次の式で与えられる。

cos x *cosh b -i*sin a *sinh b

ただし，a＋i＊b＝x。

COSD(x)　　数学

COSDは，与えられた値xの cosの値(cos x)を表わす浮動小数点数値を返す。

x　　度を単位とする値をもつ式。xは実数でなければならない。

COSH(x)　　数学

COSHは，与えられた値xの cosh の値(cosh x)を表わす浮動小数点数値を返す。

xが複素数の場合，結果は次の式で与えられる。

cosh a *cos b +i*sinh a *sin b

ただし，a＋i＊b＝x　である。

COUNT(x)　　STREAM入出力

COUNTは，指定されているファイルxに対して最後に行なわれたGETまたはPUT操作の間に転送されたデータ項目の数を表わす，省略時精度の固定小数点2進整数を返す。

x　　ファイル式。このファイルはSTREAM 属性をもつものでなければならない。

次の点に注意すること。GETまたはPUT操作の間にオン・ユニットないしプロシージャーに入り，しかもそのオン・ユニットないしプロシージャーの中で同じファイルに対してGETまたはPUT操作が行なわれている場合，COUNTの値は新しく行なわれた操作に関する値にリセットされる。この後，当初のGETまたはPUTが再開されるとCOUNTの値が復元される。

DATAFIELD　　条件取扱い

DATAFIELDは，NAME条件のためのオン・ユニットにおいて，この条件を引き起す原因となったフィールドの内容を値とする文字ストリングを返すのに使用される。

また，NAME条件に対処するためのシステム標準処置の一部として引き起されたERROR条件ないしFINISH 条件のオン・ユニットの中でDATAFIELDを使用することもできる。

文脈から外れたところでDATAFIELD が使用されていると，ナル・ストリングが返される。

DATE

DATEは，長さが6文字でyymmdd という形をとる文字ストリングを返す。ここで，

yy　今年の年号を表わす
mm　今月は何月かを表わす
dd　今日は何日かを表わす

DECIMAL (x_1 ,〔, x_2 〔, x_3 〕〕)　　算術

省略形：　DEC(x_1 〔, x_2 〔, x_3 〕〕)

DECIMALは，与えられた値 x_1 の10進数表現(x_2 および x_3 で指定されている精度をもつ)を返す。

x_1　10を基数とする表現に変換すべき値。

x_2　変換操作の間維持すべき桁数を指定するための無符号の10進整数定数。x_2 は当コンパイラーのもとでの許容限度を越えていてはならない。

x_3　結果のスケール因数を指定するための10進整数定数（符号はつけてもつけなくてもよい）。結果が固定小数点数の場合，x_2 を与えたときは x_3 も与えなければならない。結果が浮動小数点数の場合は，x_2 しか与えることができない。

x_2 および x_3 を共に省略した場合には，結果の精度は基数変換規則に従って決定される。

DIM（x_1，x_2）　配列取扱い

DIMは，与えられた配列 x_1 に関して指定されている次元 x_2 の現在の範囲を表わす省略時の精度の固定小数点2進整数を返す。

x_1　与えられた配列。この配列は現時点で割り振られているものでなければならない。

x_2　x_1 のある特定の次元を指定するための要素式。必要に応じて，x_2 は精度（15,0）の2進整数に変換される。

x_1 の次元数は，x_2 によって指定されている次元より小さいものであってはならない。

DIVIDE（x_1，x_2，x_3〔，x_4〕）　算術

DIVIDEは，2つの値 x_1 と x_2 の商を x_3 および x_4 で指定されている精度をもつ数として返す。

x_1　被除数。

x_2　除　数。

x_3　除算の間確保すべき桁数を指定するための無符号の10進整数定数。x_3 は当コンパイラーのもとでの許容限度を越えていてはならない。

x_4　結果のスケール因数を指定するための10進整数定数（符号はつけてもつけなくてもよい）。結果が固定小数点数の場合は，x_3 と x_4 の両方を与えなければならない。結果が浮動小数点数の場合は，x_3 しか与えることができない。

EMPTY　記憶域制御

EMPTYは，ゼロ・エクステントの記憶域を返す。EMPTYは，ある1つの記憶域の中のすべての割振りを解放するのに使用される。この関数の値は，あるエリア変数が割り振られるとき自動的にそのエリア変数に割り当てられるという点に注意されたい。

ERF（x）　数学

ERFは，与えられた値xの誤差関数を表わす浮動小数点数値を返す。xは実数でなければならない。

結果は次の式で与えられる。

$$\mathrm{ERF}(x) = (2/\sqrt{\pi})\int_0^x e^{-t^2}\,dt$$

ERFC（x）　数学

ERFCは，与えられた値xの誤差関数の補数を表わす浮動小数点数値を返す。xは実数でなければならない。

結果は，関数ERFを使って次のように定義される。

1－ERF（x）

EXP（x）　数学

EXPは，自然対数の底eを与えられたべき指数xだけ累乗した値を表わす浮動小数点数値を返す。

FIXED（x_1〔，x_2〔，x_3〕〕）　算術

FIXED は，与えられた値 x_1 の固定小数点数表現（x_2 および x_3 で指定されている精度をもつ）を返す。

x_1　固定小数点記数法による表現に変換すべき値。

x_2　結果の桁の総数を指定するための無符号の10進整数定数。

x_3　結果のスケール因数を指定するための10進整数定数（符号をつけてもつけなくてもよい）。x_3 を省略した場合，スケール因数はゼロとみなされる。

x_2 と x_3 を共に省略した場合，省略時の精度がとられる。結果が

2進数の場合の省略時の精度は（15,0）であり，結果が10進数の場合の省略時の精度は（5，0）である。

FLOAT(x_1 〔, x_2 〕)　　算術

FLOAT は，与えられた値 x_1 の浮動小数点数表現（ x_2 で指定されている精度をもつ）を返す。

x_1　浮動小数点記数法による表現に変換すべき値。

x_2　結果の桁の総数を指定するための無符号の10進整数定数。x_2 を省略した場合，省略時の値がとられる。これは，結果が2進数のときは21であり，結果が10進数のときは6である。

FLOOR(x)　　算術

FLOOR は，与えられた値x以下の最大の整数を返す。xは実数でなければならない。

xが精度（p，q）の固定小数点数である場合，結果の精度は次の式で与えられる。

(MIN(N,MAX(p−q+1, 1), 0)

ここで，Nは許容される最大の桁数である。

HBOUND(x_1 , x_2)　　配列取扱い

HBOUND は，与えられた配列 x_1 に関して指定されている次元 x_2 の現在の上限を表わす省略時の精度をもつ固定小数点2進整数を返す。

x_1　与えられた配列。この配列は現時点において割り振られていなければならない。

x_2　x_1 のある特定の次元を指定するための要素式。必要に応じて，x_2 は精度（15,0）の2進整数に変換される。

x_1 の次元数は，次元 x_2 より小さいものであってはならない。

HIGH(x)　　ストリング取扱い

HIGHは，各々の文字が照合順序における最高位の文字（16進法で表わすとFF）である長さxの文字ストリングを返す。

x　長さを指定するための式。必要に応じて，xは精度（15，0）の2進整数に変換される。

IMAG(x)　　算術

IMAG は，与えられた複素数値xの虚数部を返す。結果のモードは実数である。

IMAG擬似変数

この擬似変数は，ある実数値またはある複素数値の実数部を，与えられた複素変数xの虚数部に割り当てる。xのモードはCOMPLEXでなければならない。

INDEX(x_1, x_2)　　ストリング取扱い

INDEX は，ストリング x_1 の中でストリング x_2 と等しいサブストリングが開始される位置を表わす，省略時の精度をもつ固定小数点2進整数を返す。

x_1　探索されるストリング

x_2　探索の目的であるストリング

x_1 の中に x_2 が見あたらない場合には，ゼロが返される。

x_1 の中に x_2 が2つ以上ある場合には，最初の x_2 の開始位置が返される。

x_2 が配列の場合，x_1 は x_2 と同じ限界をもつ配列でなければならない。

いずれかの引数が文字または10進数である場合，この引数は文字ストリングに変換される。あるいは，引数がビット・ストリングと2進数，または2進数と2進数である場合は，これらの引数はビット・ストリングに変換される。

LENGTH(x)　　ストリング取扱い

LENGTH は，与えられたストリングxの現在の長さを表わす省略時の精度の固定小数点2進整数を返す。xが2進数の場合，xはビット・ストリングに変換される。これ以外のモードの場合，xは所要の変換を受けて文字ストリングになる。

LBOUND(x_1 , x_2) 配列取扱い

LBOUND は，与えられた配列 x_1 について指定されている次元 x_2 の現在の下限を表わす省略時の精度の固定小数点２進整数を返す。

x_1 与えられた配列。この配列は現時点において割り振られていなければならない。

x_2 x_1 のある特定の次元を指定するための要素式。必要に応じて，x_2 は精度（ 15,0）の２進整数に変換される。

x_1 の次元数は，次元 x_2 より小さいものであってはならない。

LINENO（x） STREAM入出力

LINENO は，指定されているファイルｘの現在の行番号を表わす省略時の精度の固定小数点２進整数を返す。

x ファイル式。このファイルはPRINT 属性をもつものでなければならない。

LOG（x） 数学

LOGは，与えられた値ｘの自然対数（ｅを底とする）を表わす浮動小数点数値を返す。ｘが実数の場合，ｘはゼロより大きくなければならない。ｘが複素数の場合，ｘは０＋０Ｉに等しいものであってはならない。ｘが複素数の場合，この関数の値は複数個存在するので主値だけが返される。主値は次の形をとる。

a ＋ i ＊ b

ここで，ｂは次の範囲にある。

$-\pi < b <= \pi$

LOG2（x） 数学

LOG２は，与えられた値ｘの２進対数（２を底とする）を表わす浮動小数点数値を返す。ｘは実数でなければならず，しかもゼロより大きくなければならない。

LOG10（x） 数学

LOG１０ は，与えられた値ｘの常用対数（１０を底とする）を表わす浮動小数点数値を返す。ｘは実数でなければならずしかもゼロより大きくなければならない。

LOW（x） ストリング取扱い

LOWは，各々の文字が照合順序の最下位の文字（１６進法で表わすと００）である長さｘの文字ストリングを返す。必要に応じて，ｘは精度（ 15,0）の２進整数に変換される。

x 長さを指定するための式。

MAX(x_1 , x_2 , ･･･, x_n) 算術

MAXは，１つの組を構成する２つ以上の引数の中で最大の値をもつ引数を探し，その引数の値を返す。

x_1 , x_2 , ･･･, x_n 返される最大値がその中から選ばれる値のリスト。

許容される引数の数は最高６４である。引数はすべて実数でなければならない。

引数が次のような精度の固定小数点数であるものとすると，

$(p_1, q_1), (p_2, q_2), \cdots, (p_n, q_n)$

結果の精度は次の式で与えられる。

$$(\mathrm{MIN}(N, \mathrm{MAX}(p_1-q_1, p_2-q_2, \cdots, p_n-q_n)+\mathrm{MAX}(q_1, q_2, \cdots, q_n)), \mathrm{MAX}(q_1, q_2, \cdots, q_n))$$

所要の変換が行なわれた後の引数が浮動小数点数で，その精度が p_1 , p_2 , p_3 ･･･ p_n であるものとすると，結果の精度はMAX（ p_1 , p_2 , p_3 ･･･ p_n ）である。

MIN(x_1 , x_2 , ･･, x_n) 算術

MINは，１つの組を構成する２つ以上の引数の中で最小の値をもつ引数を探し，その引数の値を返す。

x_1 , x_2 , ･･, x_n 返される最小値がその中から選ばれる値のリスト。

許容される引数の数は最高６４である。引数はすべて実数でなければならない。

引数が次のような精度の固定小数点数であるものとすると，

$$(p_1, q_1), (p_2, q_2), \cdots, (p_n, q_n)$$

結果の精度は次の式で与えられる。

$$(MIN(N, MAX(p_1-q_1, p_2-q_2, \cdots, p_n-q_n)+MAX(q_1, q_2, \cdots, q_n)), MAX(q_1, q_2, \cdots, q_n))$$

所要の変換が行なわれた後の引数が浮動小数点数であり，その精度が $p_1, p_2, p_3, \cdots, p_n$ であるものとすると，結果の精度は $MAX(p_1, p_2, p_3, \cdots, p_n)$ である。

MOD(x_1, x_2) 算術

MODは，次の式を満足する最小の正の値Rを返す。

$$(x_1 - R) / x_2 = n \quad (n\text{は整数})$$

したがって，結果は，与えられた値 x_1 が与えられた値 x_2 で割り切れるようにするために，x_1 から減ずべき最小の正の値である。

x_1 は実数でなければならない。x_1 が正の場合，結果は x_1 を x_2 で割った余りである。x_1 が負の場合，結果はこの余りに x_2 を加えた値である。

x_2 は実数でなければならない。x_2 がゼロであると，ZERODIVIDE条件が起こる。結果が浮動小数点数の場合，その精度は x_1 と x_2 の精度のうち高い方の精度になる。結果が固定小数点数の場合，その精度は次の式で与えられる。

$$(MIN(N, p_2 - q_2 + MAX(q_1, q_2)), MAX(q_1, q_2))$$

ここで，(p_1, q_1)，(p_2, q_2) はそれぞれ x_1 と x_2 の精度である。

x_1 と x_2 が異なるスケール因数をもつ固定小数点数の場合，結果の左側の方が切り捨てられることがある。この切捨てが行なわれた場合に，SIZE 条件が解禁状態にあるとこの条件が引き起される。

MULTIPLY $(x_1, x_2, x_3[, x_4])$ 算術

MULTIPLYは，2つの値 x_1 と x_2 の積（x_3 および x_4 で指定されている精度をもつ）を返す。

x_1，x_2 掛け合わせるべき値。

x_3 乗算の間確保すべき桁数を指定するための無符号の１０進整数定数。x_3 は当コンパイラーのもとでの許容限度を越えてはならない。

x_4 結果のスケール因数を指定するための１０進整数定数（符号はつけてもつけなくてもよい）。結果が固定小数点数の場合は，x_3 と x_4 の両方を与えなければならない。結果が浮動小数点数の場合は，x_3 しか与えることができない。

NULL 記憶域制御

NULLは，ナル・ポインター値，すなわち，変数のどの世代も指していない値，を返す。この組込み関数の値をあるオフセット変数に割り当てることにより，ナル・ポインター値をOFFSET に変換することができる。

OFFSET(x_1, x_2) 記憶域制御

OFFSETは，与えられた記憶域 x_2 を基準にして与えられたポインター x_1 から得られたオフセット値を返す。x_1 がナルの場合，ナル値が返される。

x_1 ポインター式。この式は記憶域 x_2 内の基底付変数のある世代を識別するものでなければならない。

x_2 エリア変数。

x_1 が要素式である場合，x_2 は要素変数でなければならない。

ONCHAR 条件取扱い

ONCHARは，CONVERSION 条件が引き起される原因となった文字を返す。CONVERSION 条件のためのオン・ユニットの中でONCHARを使用することができる。また，CONVERSION 条件に対処するためのシステム標準処置として引き起されたERROR条件ないしFINISH条件のオン・ユニットの中でONCHARを使用することもできる。

文脈からはずれたところでONCHAR組込み関数が使用されている場合は，ブランクが返される。ただし，文脈からはずれたところでONCHAR 擬似変数に割当てが行なわれており，このためにONCHAR にある値が与えられている場合は別で，この場合には，この擬似変数に割り当てられている文字がこの組込み関数によって返される。

ONCHAR擬似変数

この擬似変数は，ONCHAR 組込み関数の現在の値をリセットする。この擬似変数に割り付けられている値は長さ１の文字ストリングに変換される。再度変換が試みられたときは，新しい文字が使用される。

この擬似変数が文脈の外で使用されており，しかもONCHAR組込み関数への次の参照も文脈の外で行なわれている場合は，この擬似変数に割り当てられている文字が返される。これ以外の文脈外割当ては無視される。

ONCODE　条件取扱い

ONCODE は，当該オン・ユニットを働かせる原因となった割込みのタイプを表わす省略時の精度の固定小数点２進整数を返す。どのオン・ユニットにおいてもONCODE を使用することができる。「H節： ON条件」ですべてのONコードを定義する。

文脈の外でONCODE が使用されている場合は，ゼロが返される。

ONCOUNT　条件取扱い

ONCOUNT は，あるオン・ユニットに入ったときにまだ残っている割込みの数を表わす省略時の精度の固定小数点２進整数を返す。RECORD条件ないしTRANSMIT条件のためのオン・ユニット，または，これらの条件に対処するためのシステムの標準処置として引き起されたERROR条件ないしFINISH条件のオン・ユニットの中でしかONCOUNT を使用してはならない。というのは，複数の割込みを引き起こし得るのは，RECORD条件とTRANSMIT条件だけであるからである（「H節： ON条件」を参照のこと）。

文脈の外でONCOUNTが使用されている場合は，ゼロが返される。

ONFILE　条件取扱い

ONFILEは，入出力条件またはCONVERSION条件が提起されているファイルの名前を値とする文字ストリングを返す。入出力条件ないしCONVERSION条件のためのオン・ユニット，または，入出力条件ないしCONVERSION条件に対処するためのシステム標準処置として引き起されたERROR 条件ないしFINISH 条件のオン・ユニットの中でONFILE を使用することができる。

文脈の外でONFILE が使用されている場合は，ナル・ストリングが返される。

ONKEY　条件取扱い

ONKEY は，入出力条件を引き起す原因となったレコードのキーを値とする文字ストリングを返す。ENDFILEを除く任意の入出力条件のためのオン・ユニット，または，入出力条件に対処するためのシステムの標準処置として引き起されたERROR 条件ないしFINISH 条件のオン・ユニットの中で，ONKEY を使用することができる。KEYED ファイルにおける操作に対しては，つねにONKEY がセットされる（入出力条件を提起したステートメントでKEY，KEYTO，またはKEYFROMオプションが指定されていないときでも）点に注意されたい。

結果は以下の規則に従って決定される。

1. （ENDFILEを除く）任意の入出力条件，または，これらの条件に対処するためのシステムの標準処置として引き起されたERROR 条件ないしFINISH 条件の場合，結果はエラーを引き起こした入出力ステートメントの記録されたキーの値になる。

   REGIONAL（1）データ・セットの場合には，結果は8バイトからなる文字表現の区域番号になる。キーが間違って指定されている場合，結果は原始キーの最後の8バイトになる。原始キーが8バイトより短い場合，原始キーは右側でブランクが補われて8バイトになる。

2. 更新されたレコードのキーと入力レコードのキーとが合致しないときに更新されたレコードを索引データ・セットに書き込もうとするためにREWRITE ステートメントを用いた場合には，結果は入力レコードの内蔵されたキーの値になる。

文脈の外でONKEY が使用されている場合，ナル・ストリングが返される。

ONLOC　条件取扱い

ONLOC は，あるプロシージャーの中で当該条件が引き起された場合に，そのプロシージャーの入口点の名前を値とする文字ストリングを返す。ONLOCは，どのオン・ユニットの中でも使用することができる。

文脈の外でONLOC が使用されている場合は，ナル・ストリングが返される。

ONSOURCE　条件取扱い

ONSOURCEは，CONVERSION条件が引き起されたときに処理されていたフィールドの内容を値とする文字ストリングを返す。

CONVERSION条件のためのオン・ユニット，または，CONVERSION条件に対処するためのシステム標準処置として引き起されたERROR 条件ないしFINISH 条件のオン・ユニットの中でONSOURCEを使用することができる。

文脈の外でONSOURCE が使用されている場合は，ナル・ストリングが返される。

ONSOURCE擬似変数

この擬似変数は，ONSOURCE 組込み関数の現在の値をリセットする。この擬似変数に割り当てられている値は，文字ストリングに変換され，必要に応じて右側にブランクが補われて当該エラーを引き起こしたフィールドの長さと同じ長さをもつようになる。再び変換が試みられるときは，この新しいストリングが使用される。

再び変換が試みられるとき，この擬似変数に割り当てられているストリングは，単独のデータ項目として処理される。このような理由から，GET LISTまたはGET DATAステートメントを実行している間に変換が行なわれる場合は，エラー訂正手順の中で，2つ以上のデータ項目を含むストリングを割り当ててはならない。このストリングの中にブランクまたはコンマがあると，さらに変換エラーが起こる。

POINTER(x_1 , x_2) 記憶域制御

省略形： PTR(x_1, x_2)

POINTER は，与えられたオフセット値x_1 と与えられた記憶域x_2 とから得られたポインター値を返す。x_1 がナルの場合は，ナル値が返される。

x_1 オフセット式。この式は基底付変数のある世代を識別するものでなければならない。ただし，この基底付変数は必ずしもx_2 の中になくてもよい。この基底付変数がx_2 の中にない場合は，その世代がx_2 の世代に等しくなければならない。

x_2 エリア変数。

異なる記憶域にある基底付変数の世代が等しいということは，一番新しい世代の割振りまで，双方の変数が同じ回数だけ割り振られたり，あるいは解放されたりしたことを意味する。

POLY(x_1 , x_2) 配列取扱い

POLYは，与えられた2つの1次元配列x_1 およびx_2 から作られた多項式の値を表わす浮動小数点数値を返す。

x_1 a(m:n)として定義されている配列。ここで(m:n)は下限と上限を表わす。

x_2 x(p:q)として定義されている配列。ここで(p:q)は下限と上限を表わす。

一方の配列または双方の配列の要素が浮動小数点数でない場合には，これらの要素は浮動小数点数に変換される。

返される値は次のように定義される。

$$a(m) + \sum_{j=1}^{n-m} (a(m+j) * \prod_{i=0}^{j-1} x(p+i))$$

(q−p)<(n−m−1)ならば，(p+i)>q のとき x(p+i)=x(q)。

m=nならば，結果はa(m)。

x_2 が要素式の場合，x_2 は1つの要素からなる配列，すなわち，x(1) であるとみなされる。このとき，結果は次のように定義される。

$$\sum_{j=0}^{n-m} a(m+j) * x(1) ** j$$

x_1 は，iSUBで定義されたものであってはならない。

PRECISION(x_1 , x_2 〔, x_3〕) 算術

省略形： PREC(x_1 , x_2 〔, x_3 〕)

PRECISION は，与えられた値x_1 をx_2 とx_3 で指定されている精度の値として返す。

x_1 精度を変更すべき値。

x_2 この操作を行なっている間確保すべき桁数を指定するための無符号の10進整数定数。x_2 が当コンパイラーのもとでの許容限度を越えていてはならない。

x_3 結果のスケール因数を指定するための10進整数定数（符号はつけてもつけなくてもよい）。結果が固定小数点数の場合，x_3 を与えなければならない。結果が浮動小数点数の場合，x_3 を与えてはならない。

PROD(x) 配列取扱い

PRODは，与えられた配列xの中のすべての要素の積を返す。

x　整数または浮動小数点数の要素からなる配列。

xの要素が整数でない固定小数点数の場合は，これらの要素は浮動小数点数に変換される。

xの要素がストリングの場合，これらの要素は整数に変換される。結果が固定小数点整数の場合，結果の精度は（N，0）である。ここで，Nは許容される最大の桁数である。xはiSUBで定義されたものであってはならない。

REAL(x)　算　術

REALは，与えられた複素数値xの実数部を返す。xが実数の場合，xは複素数に変換される。

REAL擬似変数

この擬似変数は，与えられた複素変数xの実数部に実数値または複素数値の実数部を割り当てる。xは実数であってはならない。

REPEAT(x_1，x_2)　ストリング取扱い

REPEATは，x_2 で指定されている回数だけストリングx_1 を連結したストリングをストリングx_1 に返す。したがって，結果として返されるストリングは，（x_1 ＋1）個のストリングx_1 から構成されたものになる。

x_1　繰返し用いられるストリング。

x_2　繰返しの回数を表わす整数に変換することのできる式。

x_2 がゼロまたは負の場合，ストリングx_1 が返される。

x_2 が配列の場合，x_1 はx_2 と同じ限界をもつ配列でなければならない。

x_1 が算術データの場合，x_1 はストリングに変換される。x_1 が2進数のときは，x_1 はビット・ストリングに変換され，x_1 が10進数のときは，x_1 は文字ストリングに変換される。

ROUND(x_1，x_2)　算　術

ROUNDは，与えられた値x_1 をx_2 で指定されている桁でまるめた値を返す。

x_1　まるめを行なう値。

x_2　まるめを行なうべき桁を指定するための10進整数定数（符号をつけてもつけなくてもよい）。x_2 が正の場合，小数点から右にx_2 番目の桁がまるめられる。x_2 が負の場合，小数点から左に（x_2 ＋1）番目の桁がまるめられる。

x_1 が浮動小数点数の場合，x_2 は無視され，小数部の右端のビットが1にセットされる。

結果が固定小数点数の場合の精度は，次の式で与えられる。

(MAX(1, MIN(p−q+1+x_2, N)), x_2)

ここで，（p，q）はx_1 の精度であり，Nは許容される最大の桁数である。

負の値をまるめる場合は，まずその絶対値をまるめこの後符号を取り替えるという点に注意されたい。

SIGN(x)　算　術

SIGNは，与えられた値xが正，ゼロ，または負のいずれであるかを示す省略時の精度の固定小数点2進整数を返す。返される値を以下に示す。

x の 値	返される値
x＞0	＋1
x＝0	0
x＜0	−1

xは実数でなければならない。

SIN(x)　数　学

SINは，与えられた値xのsin の値（sin x）を表わす浮動小数点数値を返す。

x　ラジアン単位の値をもつ式。

xが複素数の場合，結果は次の式で与えられる。

sin(a)*cosh(b)+i*cos(a)*sinh(b)

ここで，a＋i＊b＝x　である。

SIND(x)　数　学

SINDは，与えられた値xのsinの値（sin x）を表わす浮動小数点数値を返す。

x　度単位の値をもつ式。xは実数でなければならない。

SINH(x)　数学

SINHは，与えられた値xのsinhの値（sinh x）を表わす浮動小数点数値を返す。xが複素数である場合，結果は次の式で与えられる。

sinh(a)*cos(b)+i*cosh(a)*sin(b)

ここで，a＋i＊b＝x。

SQRT(x)　数学

SQRTは，xの平方根を表わす浮動小数点数値を返す。xが実数である場合，xはゼロより小さいものであってはならない。結果はxの正の平方根である。xが複素数である場合，この関数は複数個の値をもつので，その主値だけが返される。主値は次のような形をとる。

（a＋i＊b）

ここで，a＞0，または，a＝0かつb＞＝0である。

STATUS(x)　非同期入出力

STATUSは，与えられた事象xの状況値を表わす省略時の精度の固定小数点2進整数を返す。事象が正常である場合にはゼロが返され，異常である場合にはゼロ以外の値が返される。

STATUS擬似変数

この擬似変数は，与えられた事象xの状況変数をリセットする。この擬似変数への割当てが行なわれている間は如何なる割込みも起こらない。

STRING(x)　ストリング取扱い

STRINGは，あるストリング・データ集合体の全要素を連結して得られた1つの要素ストリングを返す。

x　要素が，すべて，文字ストリングと数字データ，または，文字ストリングか数字データ，または，ビット・ストリング，である配列式ないし構造式。

xは，演算式あるいは関数参照であってはならない。

xがALIGNED属性の要素の性質上パディングが行なわれている構造である場合，このパディングは結果の中に含まれない。

ストリング・データ集合体の中のいずれかのストリングが長さの変わるストリングである場合，2バイトの長さ接頭語を除く現在の長さだけが連結される。xは，iSUBで定義されたものであってはならない。

xが要素変数である場合にも，ストリング・データ集合体に関して記述した規則が適用される。ただし，この場合連結は行なわれない。

STRING擬似変数

この擬似変数は，与えられた集合体変数xに，ストリングを1つずつ割り当てる。この割り当ては，集合体の要素すべてが満たされるまで，あるいは割り当てられるストリングがなくなるまで，続けられる。後者の場合，集合体変数の中の未だ割当ての行なわれていないストリングにブランクが満たされるか，あるいは，長さの変わるストリングについてはこれらに長さゼロが与えられる。

GETステートメントのデータ指定の中やREADステートメントのINTOまたはKEYTOオプションの中でSTRING擬似変数を使用してはならない。

長さの変わるストリングはその最大の長さまで満たされる。

SUBSTR(x_1, x_2〔, x_3〕)　ストリング取扱い

SUBSTRは，与えられたストリングx_1のあるサブストリングを返す。

x_1　当該サブストリングが抽出されるストリング。

x_2　x_1における当該サブストリングの開始位置を示すための整数に変換することのできる式。

x_3　x_1の中の当該サブストリングの長さを指定するための整数に変換することのできる式。x_3がゼロの場合，ナル・ストリングが返される。x_3を省略した場合，返されるサブストリングはx_1の位置x_2から始って終りに至る部分である。

x_1がストリングでない場合，x_1が2進数であれば，x_1はビット・ストリングに変換され，x_1が10進数ならば，x_1は文字ストリングに変換される。

当該サブストリングがx_1の中に存在し得ないようなx_2およびx_3の値が指定されている場合には，STRINGRANGE条件が解禁状態にあればこの条件が引き起される。STRINGRANGE条件が解禁条件になければ，この場合の結果はどうなるかわからない。

SUBSTR擬似変数

この擬似変数は，与えられたストリングxのサブストリングに，あるストリングを割り当てる。このストリングの他の部分は何の変更も受けない。

SUM(x)　　配列取扱い

SUMは，与えられた配列xの中のすべての要素の和を返す。

x　　算術要素からなる配列。

xの要素が固定小数点数の場合，結果の精度は(N，q)である。ここで，Nは許容される最大の桁数であり，qはxのスケール因数である。xの要素がストリングの場合，これらの要素は整数に変換される。xはiSUBで定義されたものであってはならない。

TAN(x)　　数学

TANは，与えられた値xのtanの値(tan x)を表わす浮動小数点数値を返す。

x　　ラジアン単位の値をもつ式。

xが複素数の場合，結果は次のように定義される。

REAL (TAN(x))=SIN(2*a)/(COS(2*a)+COSH(2*b))

IMAG (TAN(x))=SINH(2*b)/(COS(2*a)+COSH(2*b))

ここで，a+i*b=x。

TAND(x)　　数学

TANDは，与えられた値xのtanの値(tan x)を表わす浮動小数点数値を返す。

x　　度単位の値をもつ式。xは実数でなければならない。

TANH(x)　　数学

TANHは，与えられた値xのtanhの値(tanh x)を表わす浮動小数点数値を返す。

x　　ラジアン単位の値をもつ式。

xが複素数の場合，結果は次のように定義される。

-i*TAN(i*x)

TIME

TIMEは，hhmmsstttという形の長さ9の文字ストリングを返す。ここで，

hh	現時点の時間
mm	分
ss	秒
ttt	ミリセカンド

計時機構が使用可能でない場合，TIMEは(9)'0'と言う値を返す。

TRANSLATE(x_1, x_2〔, x_3〕)　　ストリング取扱い

TRANSLATEは，与えられたストリングx_1と同じ長さをもち，x_1の一部またはすべての文字が他の文字と置き換えられているストリングを返す。x_1の中の文字は，ストリングx_2およびx_3が与える翻訳テーブルに従って変更される。

TRANSLATEは，x_1の中の個々の文字を次のように処理する。

x_1の中のある文字がx_3の中にある場合には，x_1の中のこの文字に対応するx_2の文字が結果として複写される。これ以外の場合には，x_1の中の文字が結果として直接複写される。

x_1　　翻訳のもとになる文字ストリング。

x_2　　文字の翻訳値(翻訳された後の値)を含んでいる文字ストリング。

x_3　　翻訳すべき文字を含んでいる文字ストリング。x_3を省略した場合，x_3は256文字からなるストリングであるとみなされる。このストリングにはすべての文字(16進法で表わすと00からFF)が昇順に並んでいる。

ストリングx_2とストリングx_3の長さは，同じでなければならない。両者の長さが異なる場合，x_2の長さをx_3の長さと等しくするために，x_2の右側にブランクが補充されるか，またはx_2の右端が切り捨てられる。

文字以外の引数はすべて文字に変換される。

TRUNC(x) 配列取扱い

TRUNC は，与えられた値xに対して切捨てを行なった後の整数を返す。xが正またはゼロである場合，結果はx以下の最大の整数である。xが負の場合，結果はx以上の最小の整数である。xは実数でなければならない。

xが精度（p，q）の固定小数点数であるとすると，結果の精度は次の式で与えられる。

(MIN(N,MAX(p−q+1，1))，0)

ここで，Nは許容される最大の桁数である。

UNSPEC(x) ストリング取扱い

UNSPEC は，与えられた値xの内部コード形式であるビット・ストリングを返す。

x　任意のデータ・タイプの式。

返されるビット・ストリングの長さはxの属性によって決まる。

xが複素数の場合，返されるストリングの長さは下の表に示した値の2倍である。

ビット・ストリングの長さ	x の 属 性
16	FIXED BINARY (p,q) ただし，p<16
32	FIXED BINARY (p,q),ただし p>15 FLOAT BINARY (p)， ただし p>22 FLOAT DECIMAL(p)， ただし p<7 POINTER OFFSET FILE 定数または変数
64	FLOAT BINARY (p)，ただし 21<p<54 FLOAT DECIMAL(p)，ただし 6<p<17 LABEL 定数または変数 ENTRY 定数または変数
256	EVENT

ビット・ストリングの長さ	x の 属 性
n	BIT (n)
n+16	BIT VARYING ここで，組込み関数の場合nは現在の長さであり，擬似変数の場合nは最大の長さである。
8＊n	CHARACTER (n) PICTURE （文字ストリングの長さはn）
8＊(n+2)	CHARACTER (n) VARYING
8＊(n+16)	AREA (n)
8＊FLOOR(n)	FIXED DECIMAL (p,q) ここで，n＝(p＋2)／2

UNSPEC擬似変数

この疑似変数は，与えられた変数xにビット・ストリングを直接割り当てる。つまり，変数のデータ・タイプの変換は試みられないわけである。ビット・ストリングの長さを変数の長さと等しくするために，必要に応じて，ビット・ストリングの右側にゼロが補充される。

xが長さの変わるストリングの場合，その2バイトの接頭語もビット・ストリングが割り当てられるフィールドの中に入る。

VERIFY(x_1，x_2) ストリング取扱い

VERIFY は，与えられたストリングx_1の中で，与えられたストリングx_2にない文字またはビットが出て来た最初の位置を示す省略時の精度の固定小数点2進整数を返す。x_1の中にある文字またはビットがすべてx_2の中にもある場合には，ゼロという値が返される。引数が2つとも算術データの形になっている場合，これらの引数はストリングに変換される。一方の引数がビット・ストリングで他方の引数が文字ストリングの場合，ビット・ストリングが文字ストリングに変換される。

x_1　x_2の中にない文字を求めて走査されるストリング。

x_2　任意の順序に並んだ1組の文字からなる検査ストリング。

いずれかの引数が文字または10進数である場合，この引数は文字ストリングに変換される。これ以外には，引数がビット・ストリングと2進数の場合，または双方の引数が2進数の場合，変換が行なわれてこれらの引数はビット・ストリングになる。

H節： ON条件

ON条件は，PL／IのONステートメントで指定することのできる例外条件である。ある条件が解禁状態にあるとき，その条件が引き起されると割込みが起こる。割込みが起こると，その条件に対処すべく現時点において指定されている処置がとられる。その条件を対象とするONステートメントが効力をもっていない場合には，現時点で指定されている処置はその条件を対象とするシステム標準処置である。その条件を対象とするONステートメントが効力をもっている場合，現時点で指定されている処置は，SYSTEM（その条件を対象とするシステムの標準処置がとられる）またはあるオン・ユニット（プログラマーがその条件に対処するための独自の処置を指定したもの）である。

ある条件が解禁状態にない場合（つまり，その条件が禁止状態にある場合），コンパイラーは，実行中にその条件をテストするためのコードを作り出さない。この条件が引き起された場合エラーが発生するであろうが，この条件によってシステム割込みが引き起こされない限り実行が停止されることはない。

例外条件の中には，条件接頭語によって明示的に禁止状態に置かれない限りつねに解禁状態にある条件がある。また，条件接頭語によって明示的に解禁状態に置かれない限りつねに禁止状態にある条件がある。さらに，つねに解禁状態にあり，禁止状態に置くことのできない条件もある。

条件接頭語によって明示的に禁止状態に置かれない限りつねに解禁状態にある条件を以下に示す。

CONVERSION

FIXEDOVERFLOW

OVERFLOW

UNDERFLOW

ZERODIVIDE

条件名の前にNOと記入した条件接頭語を用いることによって，上に掲げた条件の各々を禁止状態に置くことができる（条件接頭語を作る際条件名とNOの間にブランクを置いてはならない）。したがって，次の各々の条件接頭語によって，対応する条件が禁止状態に置かれる。

NOCONVERSION

NOFIXEDOVERFLOW

NOOVERFLOW

NOUNDERFLOW

NOZERODIVIDE

このような条件接頭語を指定することによって，その接頭語の有効範囲の間これと対応する条件が禁止状態に置かれる。この有効範囲の外ではこの条件は依然として解禁状態にある。（条件接頭語の有効範囲については，「第１４章： 例外条件の取扱いとプログラム・チェックアウト」を参照されたい。）

上に掲げた条件とは逆に，次に示すのは条件接頭語によって解禁状態に置かれない限りつねに禁止状態にある条件である。

SIZE

SUBSCRIPTRANGE

STRINGRANGE

STRINGSIZE

CHECK

これらの条件名が条件接頭語の中に現われると，その接頭語の有効範囲の間，その条件が解禁状態に置かれる。しかし，この有効範囲の外ではこの条件は依然として禁止状態にある。また，NOSIZE，NOSUBSCRIPTRANGE，NOSTRINGRANGE，NOSTRINGSIZE，またはNOCHCKと指定している条件接頭語があると，その接頭語の有効範囲の間，対応する条件が禁止状態に置かれる。

上掲以外の条件はすべてつねに解禁状態にありプログラムを実行している間ずっとこの状態にとどまる。このような条件を以下に示す。

AREA
CONDITION
ENDFILE
ENDPAGE
ERROR
FINISH
KEY
NAME

RECORD

TRANSMIT

UNDEFINEDFILE

条件コード(ONコード)

プログラマーは，任意のオン・ユニットの中でONCODE組込み関数を使って，当該オン・ユニットに入る原因となったエラーないし条件の性格を調べることができる。当コンパイラーによってチェックされる条件およびエラーに対応するコードを以下に示す。

コード	エラーまたは例外条件
0	オン・ユニットの外部でONCODE 関数が使用されている。

ERROR条件コード

3　実行の前にすでにエラーがあると診断されていたステートメントを実行しようとした。

FINISH条件コード

4　SIGNAL FINISH，またはSTOPステートメントが実行された。
または，
主プロシージャーが正規に終了した。

ERROR条件コード

9　SIGNAL ERRORステートメントが実行された。

注：　上述以外のERROR 条件コードについては，コード番号1000の後の所を参照のこと。

NAME条件コード

10　SIGNAL NAMEステートメントが実行された。
または，
GET DATA入力流れの中に認定できない識別名がある。

RECORD条件コード

20　SIGNAL RECORDステートメントが実行された。

21　レコード・サイズの値より小さいレコード変数がある。

22　レコード・サイズの値より大きいレコード変数がある。

23　長さがゼロのレコードを書き出そうとしたり，あるいは探そうとする試みがなされたか，あるいは，小さすぎて内蔵されたキーを収容できないレコードがある(キー順のVSAMの場合)。

24　REGIONAL データ・セットから長さがゼロのレコードが読み取られた。

TRANSMIT条件コード

40　SIGNAL TRANSMITステートメントが実行された。

41　出力データ・セットの中に訂正不可能な転送エラーがある。

42　入力データ・セットの中に訂正不可能な転送エラーがある。

43　キー順のVSAM データ・セットの索引セットへの出力に訂正不可能な転送エラーがある。

44　キー順のVSAM データ・セットの索引セットからの入力に訂正不可能な転送エラーがある。

45　キー順のVSAM データ・セットの順序セットへの出力に訂正不可能な転送エラーがある。

46　キー順のVSAM データ・セットの順序セットからの入力に訂正不可能な転送エラーがある。

KEY条件コード

50　SIGNAL KEYステートメントが実行された。

51　指定されているキーが見つからない。

52　INDEXED データ・セットまたはキー順のVSAM データ・セットの中にすでに存在しているレコードのキーと同じキーをもつキー付レコードを追加しようとしたか，または，REGIONAL(1)データ・セットを作成する際に，すでにレコードが収容されている区域に書込もうとした。

53　INDEXEDデータ・セット，キー順のVSAM データ・セット，またはREGIONALデータ・セットを順次方式で作成しているときに，KEYFROM オプションで指定されている式の値が，先に指定されているキーまたは区域番号の値より小さい。

54　区域番号が数字でない，などという理由で，キー変換エラーが発生した。

55　キーの指定がナル・ストリングであるか，(8)'1'B で始まっている，あるいは，INDEXED データ・セットまたはキー順VSAM データ・セットを対象とする順次方式のREWRITE〔FROM〕において内蔵されたキーの変更が行なわれた。

56　当該データ・セットの範囲の外にあるキーを使ってレコードにアクセスしようとした。

57　キー付レコードを追加するためのスペースがない。

58　指定されているキーが当該データ・セットに対して定義されているキーの範囲の外にある（キー順VSAMのみ）。

ENDFILE条件コード

70　SIGNAL ENDFILEステートメントが実行された。
または，
ファイルの区切りを越えて読み取ろうとした。

UNDEFINEDFILE条件コード

80　SIGNAL UNDEFINEDFILE ステートメントが実行された。

81　ファイルを開く時点において，DECLARE ステートメントで指定されている属性と，明示的または暗示的な OPEN ステートメントで指定されている属性との間に，ファイルの属性の矛盾が存在する。

82　INPUT/UPDATE を行なうためのデータ・セットとして，空白のVSAMデータ・セットが開かれた。

83　環境の指定（ENV，DDカード，DSCBなど）が不完全である。

84　ASSGN，DLBL，または EXTENT ステートメントにエラーがある（VSAMのみ）。

85　あるREGIONAL データ・セットと結び付けられているOUTPUT ファイルの初期設定を行なっている間に，入出力エラーが発生した。
または，
INDEXED データ・セットを作成している場合にマスター索引用またはシリンダー索引用として与えられているスペースが充分でない。

86　行の長さが当コンパイラーにおいて定義されている最大値より長い。
または，
ENVIRONMENT オプションで有効でない値が指定されている。

87　ENVIRONMENT のオプションに矛盾がある。たとえば，レコード形式とブロック・サイズが両立し得ない。
または，
コンパイラーが，ファイル宣言の中に訂正不可能なエラーを見つけた。

89　合言葉が有効でないか，または合言葉が抜けている（VSAMのみ）。

ENDPAGE条件コード

90　SIGNAL ENDPAGEステートメントが実行された。

または，
行番号が現在のページ・サイズに等しいのに新しい行を開始しようとした。

92　VSAM データ・セットにアクセスするにあたってそのデータ・セットを開く際，オペレーティング・システムがエラーを見つけた。

93　VSAM データ・セットを開く際に，認識不可能なエラーをVSAM が見つけた。

STRINGSIZE条件コード

150　SIGNAL STRINGSIZE ステートメントが実行された。
または，
あるストリングがそれより短いストリングに割り当てられた。

OVERFLOW条件コード

300　SIGNAL OVERLOW ステートメントが実行された。
または，
浮動小数点数の大きさが許容最大値を越えている。

FIXEDOVERFLOW条件コード

310　SIGNAL FIXEDOVERFLOWステートメントが実行された。
または，
固定小数点数算術演算の結果の長さが許容最大値を越えている。

ZERODIVIDE条件コード

320　SIGNAL ZERODIVIDEステートメントが実行された。
または，
ゼロで割ろうとした。

UNDERFLOW条件コード

330　SIGNAL UNDERFLOWステートメントが実行された。
または，
浮動小数点数の大きさが許容最小値より小さい。

SIZE条件コード

340　SIGNAL SIZEステートメントが実行された。
または，
変数あるいは一時フィールドへの割当てにあたって，高位のゼロ以外の桁が失われた，あるいは，入出力操作の間に有効数字が失われた。

341　入出力操作の間に，高位のゼロ以外の桁が失われた。

STRINGRANGE条件コード

350　SIGNAL STRINGRANGEステートメントが実行された。
または，
SUBSTR参照の引数の長さがSUBSTR組込み関数の項で説明した規則に合致していない。

AREA条件コード

380　割振りを行なうための記憶域が充分にないエリアに，基底付変数を割り振ろうとした。

381　原エリアを割り当てるに充分なベースが目標エリアにない。

382　SIGNAL AREAステートメントが実行された。

CONDITION条件コード

500　SIGNAL CONDITION(条件)ステートメントが実行された。

CHECK条件コード

510　SIGNAL CHECK ステートメントが実行された。
または，
CHECK 接頭語の有効範囲において，変数のすべてまたは一部の値が変化しようとしているか，あるいはラベルないし名前が指定されたステートメントの実行が始まろうとしている。

SUBSCRIPTRANGE条件コード

520　SIGNAL SUBSCRIPTRANGE ステートメントが実行された。
または，
添字の値が計算された結果，指定された限界の外にあることがわかった。

521　iSUBで定義されている変数の添字が，基底変数の対応する次元に割り振られている限界の外にあることがわかった。

CONVERSION条件コード

600　SIGNAL CONVERSIONステートメントが実行された。

603　GET STRINGステートメントのF形式項目を処理している間にエラーがあった。

604　GET FILE ステートメントのF形式項目を処理している間にエラーがあった。

605　TRANSMIT条件の後実行されるGET FILEステートメントのF形式項目を処理している間にエラーがあった。

606　GET STRINGステートメントのE形式項目を処理している間にエラーがあった。

607　GET FILE ステートメントのE形式項目を処理している間にエラーがあった。

608　TRANSMIT条件の後実行されるGET FILEステートメントのE形式項目を処理している間にエラーがあった。

609　GET STRINGステートメントのB形式項目を処理している間にエラーがあった。

610　GET FILE ステートメントのB形式項目を処理している間にエラーがあった。

611　TRANSMIT条件の後実行されるGET FILE ステートメントのB形式項目を処理している間にエラーがあった。

612　文字ストリングから算術データへの変換の間にエラーがあった。

613　GET FILEまたはPUT FILE ステートメントで行なわれた文字ストリングから算術データへの変換の間にエラーがあった。

614　TRANSMIT条件の後実行されるGET FILEまたはPUT FILEステートメントで行なわれた，文字ストリングから算術データへの変換の間にエラーがあった。

615　文字ストリングからビット・ストリングへの変換の間にエラーがあった。

616　GET FILEまたはPUT FILE ステートメントで行なわれた文字ストリングからビット・ストリングへの変換の間にエラーがあった。

617　TRANSMIT条件の後実行されるGET FILEまたはPUT FILE ステートメントで行なわれた文字ストリングからビット・ストリングへの変換の間にエラーがあった。

618　文字ストリングからピクチャーへの変換の間にエラーがあった。

619　GET FILEステートメントで行なわれた文字ストリングからピクチャーへの変換の間にエラーがあった。

620　TRANSMIT条件の後実行されるGET FILEステートメントで行なわれた文字ストリングからピクチャーへの変換の間にエラーがあった。

621　GET STRINGステートメントのための10進P形式項目にエラーがある。

622　GET FILE ステートメントの10進P形式入力にエラーがある。

623　TRANSMIT条件の後実行されるGET FILEステートメントの10進P形式入力にエラーがある。

625　GET FILE ステートメントの文字P形式入力にエラーがある。

626　TRANSMIT条件の後実行されるGET FILEステートメントの文字P形式入力にエラーがある。

630　LIMSCONVのもとで使用可能なSTREAM 入出力変換の限定セットに含まれていない変換が，GET

STRING LISTまたはGET STRING DATA ステートメントで要求されている（STREAM入出力変換セットでは，変換の種類が制限されている）。

631　LIMSCONVのもとで使用可能なSTREAM 入出力変換の限定セットに含まれていない変換がGET FILE LISTまたはGET FILE DATA ステートメントで要求されている（STREAM入出力変換セットでは，変換の種類が制限されている）。

632　TRANSMIT条件の後実行されるGET FILE LISTまたはGET FILE DATA ステートメント(LIMSCONVのもとで使用可能なSTREAM入出力変換の限定セットを使用している）で行なわれた変換の間にエラーがあった。

ERROR条件コード

注：　以下に記載されているもの以外のERROR 条件については，条件コード9を参照のこと。

1002　GET STRINGまたはPUT STRINGで，ストリングの長さより長いデータが指定されている。

1003　当該データ・セットに関連して先に引き起されていたTRANSMIT 条件またはKEY条件のために，出力が停止された。

1004　印刷用ファイル以外のファイルに対して，PAGE，LINE，またはSKIP≦0 を使用しようとした。

1005　DISPLAY（要素式）REPLY（文字変数）ステートメントの要素式ないし文字変数の長さがゼロである。

1007　REWRITEまたはDELETE ステートメントの直前にREADがある。

1008　GET STRING DATAステートメントで指定されているストリングの中に，認定されていない識別名がある。

1009　入出力ステートメントの中で，当該ファイルの属性と矛盾している操作ないしオプションが指定されている。

1011　データ管理プログラムによって入出力エラーが検出されたが，そのエラーの原因に関する情報を与えることができない。

1014　未完了の入出力操作の数が当コンパイラーのもとでの許容限度を越えているのに，さらに他の入出力操作を開始しようとした。

1015　すでに使用されているEVENT 変数を用いて入出力操作が指定されている。

1016　暗示的にファイルを開こうとして失敗したためにUNDEFINEDFILE 条件が引き起され，この後，オン・ユニットから制御が正常な形で返されて当該ファイルが開かれていないことがわかった。

1018　データ・リストの終りまたは（編集指示転送における）形式リストの終りに達する前に，データの中にファイルの終りないしストリングの終りが出て来た。

1022　使用可能なスペースがない順次方式の出力データ・セットにレコードを書き込もうとした。

1024　関連するファイルに入出力順序エラーがある。

1025　使用可能な仮想記憶域が不充分であるために，入出力要求を完了することができない。

1028　複数ボリュームVSAM データ・セットの取り付けられていないボリュームに存在するレコードが要求されている。

1500　計算エラー：　SQRT 組込み関数の短精度浮動小数点引数が負である。

1501　計算エラー：　SQRT 組込み関数の長精度浮動小数点引数が負である。

1504　LOG，LOG2，またはLOG10組込み関数における計算エラー：　短精度浮動小数点引数≦0。

1505　LOG，LOG2，またはLOG10組込み関数における計算エラー：　長精度浮動小数点引数≦0。

1506　SIN，COS，SIND，またはCOSD 組込み関数における計算エラー：　短精度浮動小数点引数の絶対値が，(2**18)*π（SINおよびCOS の場合）または(2**18)*180（SINDおよびCOSDの場合）を越えている。

1507　SIN，COS，SIND，またはCOSD 組込み関数における計算エラー：　長精度浮動小数点引数の絶対値が，(2**50)*π（SINおよびCOS の場合）または(2**50)*180（SINDおよびCOSDの場合）を越えている。

1508　計算エラー：　TANまたはTAND 組込み関数の短精度浮動小数点引数の絶対値が，それぞれ(2**18)*πまたは(2**18)*180を越えている。

1509　計算エラー：　TANまたはTAND 組込み関数の長精度浮動小数点引数の絶対値が，それぞれ(2**50)*πまたは(2**50)*180を越えている。

1510　計算エラー：　ATANまたはATAND 組込み関数の短精度浮動小数点引数が共にゼロである。

1511　計算エラー：　ATANまたはATAND 組込み関数の長精度浮動小数点引数が共にゼロである。

1514　計算エラー：　ATANH 組込み関数の短精度浮動小数点引数の絶対値≧1。

1515　計算エラー：　ATANH 組込み関数の長精度浮動小数点引数の絶対値≧1。

1518　計算エラー：　ASINまたはACOS 組込み関数の短精度浮動小数点引数の絶対値が1を越えている。

1519　計算エラー：　ASINまたはACOS 組込み関数の長精度浮動小数点引数の絶対値が1を越えている。

1550　計算エラー：　実数の短精度浮動小数点数である底がゼロで，その整数の指数が正でない。

1551　計算エラー：　実数の長精度浮動小数点数である底がゼロで，その整数の指数が正でない。

1552　計算エラー：　実数の短精度浮動小数点数である底がゼロで，その非整数の指数が正でない。

1553　計算エラー：　実数の長精度浮動小数点数である底がゼロで，その非整数の指数が正でない。

1554　計算エラー：　複素数の短精度浮動小数点数である底がゼロで，固定小数点整数である指数が正でない。

1555　計算エラー：　複素数の長精度浮動小数点数である底がゼロで，固定小数点整数である指数が正でない。

1556　計算エラー：　複素数の短精度浮動小数点数である底がゼロで，固定小数点指数が正でも実数でもない。

1557　計算エラー：　複素数の長精度浮動小数点数である底がゼロで，非整数の指数が正でも実数でもない。

1558　計算エラー：　ATANまたはATANH組込み関数の複素数短精度浮動小数点引数が，それぞれ±1Iまたは±1である。

3799　変換すべきストリングに有効でない文字があり，このために引き起されたCONVERSION条件の結果として入ったオン・ユニットにおいて，ONSOURCE またはONCHAR 擬似変数を使ってこの文字が訂正されていない。

3800　データ集合体の長さがシステムの許容限度（2＊＊24バイト）を越えている。

3801　構造の中にある配列のある要素をマップすることができない。

3808　COBOLまたはFORTRANで，データ集合体をマップすることができない。

3904　COMPLETION擬似変数の引数として指定されている事象変数は，あるDISPLAY ステートメントのためにすでに使用されているものである。

3905　理由のわからない入出力エラーが発生した。

3906　すでに活用状態にある事象変数への割当てがあった。

3907　すでに対応づけられている事象変数を対応づけようとした。

3911　オン・ユニットにおいて，すでにWAITステートメントの対象となっていた入出力事象を対象とするWAITステートメントが出て来た。

4001　CONTROLLED変数が割り振られる前に，その変数が参照された。

8091　命令例外。

8092　特権命令例外。

8093　EXECUTE例外。

8094　記憶保護例外。

8095　アドレシング例外。

8096　指定例外。

8097　データ例外。

9004　連係編集プログラムが，指定のデータ・セットの中に入口定数を見つけることができない。

9200　SORT/MERGEプログラムにおいてプログラム・チェックが発生した。

9201　SORT出口に対するものとして指定されているパラメーターないし返される値が，PLISRT の呼出しにおける指定と合致しない。

複数の割込み

複数の割込みとは，2つ以上の割込みが同時に発生することである。当コンパイラーでは,複数の割込みはTRANSMIT およびRECORD 両条件に起因するものだけである。複数の割込みが発生した場合，つねに，TRANSMIT に起因する割込みが最初に処理される。正常の制御の返還を行なうオン・ユニットがTRANSMIT に対して設けられていない場合は，RECORD

に起因する割込みは無視される。

条件のリスト

以下は各種条件をアルファベット順に列挙したものである。通常,各条件ごとに次に示すような情報を記載してある。

1. 各条件の一般形式を見出しとして示してある。

2. 説明 — その条件についての説明。その条件を引き起す可能性のある環境についての説明を含む。SIGNAL ステートメントを使用すれば,つねに,解禁状態にある条件を引き起すことができるという点に注意されたい。以下の説明では,この事実をそのつど取り上げることはしていない。

3. 結果 — その条件を引き起す原因となった操作の結果。このことは,その条件が解禁状態にある場合も,禁止状態にある場合も,適用することができる。場合によっては,結果が定義できないこともある。つまり,結果がどうなるか予測することができない。これに該当する場合には,その旨記してある。

4. システムの標準処置 — ある割込みが起ったときに,プログラマーの側でその割込みを処理するためのオン・ユニットを指定していない場合にシステムによってとられる処置。

5. 状況 — プログラムの実行が始まるときその条件が解禁状態にあるか禁止状態にあるかの表示。その条件を解禁状態にする方法または(可能な場合には)禁止状態にする方法についての説明を含む。

6. 正常な戻り — そのオン・ユニットを正常な形で終えた後制御が戻される場所。GO TO ステートメントでオン・ユニットの外に制御を渡すと,オン・ユニットは異常停止することになる。SIGNALステートメントによってある条件が引き起された場合は,正常の戻りであればつねにSIGNALの直後のステートメントに制御が戻されるということに注意していただきたい。

条件の分類

条件は次のように分類される。

1. 計算関係の条件 — データの取扱い,式の値の計算,および計算に関係する条件であり,次のものがある。

   CONVERSION
   FIXEDOVERFLOW
   OVERFLOW
   SIZE
   UNDERFLOW
   ZERODIVIDE

2. 入出力条件 — データの転送に関係のある条件であり,次のものがある。

   ENDFILE
   ENDPAGE
   KEY
   NAME
   RECORD
   TRANSMIT
   UNDEFINEDFILE

3. プログラム・チェックアウト条件 — プログラムのデバッグを容易にするための条件であり,次のものがある。

   CHECK
   SUBSCRIPTRANGE
   STRINGRANGE
   STRINGSIZE

4. 記憶域制御に関する条件

   AREA

5. システムの処置に関する条件 — 条件が引き起された後またはプログラムが完了した後とられるシステムの標準処置を拡張するための手段となる条件。以下のものがある。

   ERROR
   FINISH

6. プログラマーが名付ける条件

   CONDITION

AREA　　記憶域制御

説明: 次のいずれかの環境においてAREA条件が引き起される。

1. ある基底付変数が割り振られるだけの記憶域があいていないエリアにこの変数を割り振ろうとしたとき。

2. 原エリアに対する割振りを収容できるだけの記憶域が目標エリアにないのに,エリア割当てを行なおうとしたとき。

結果: いずれの場合にも,このような割振りをしようとしても割当てに何の効果もない。

システムの標準処置: オン・ユニットが与えられていない場合,

システムはメッセージを印刷し，ERROR 条件を起こす。

状況：　AREA はつねに解禁状態にあり，禁止状態に置くことはできない。

正常な戻り：　オン・ユニットからの正常な戻りの後とられる措置は，次に示す通りである。

1. 割振りによってこの条件が引き起された場合には，再びこの割振りが試みられる。ただし，この試みを行なう前に，エリア参照の値が再度計算される。したがって，オン・ユニットで小さすぎるエリアへの参照を修飾しているポインターの値を変更して，このポインターが別のエリアを指すよう修正した場合は，この割振りは，新しいエリアで行なわれるように試みられる。

2. エリア割当てないしSIGNAL ステートメントによってこの条件が引き起された場合，割込みの発生した場所から実行が続行される。

CHECK〔（名前のリスト）〕　　プログラム・チェックアウト

説明：　CHECK条件が引き起されるのは，CHECK条件接頭語の有効範囲の中だけである。この条件接頭語は，DECLARE，DEFAULT，およびENTRY ステートメント以外であればどのようなステートメントの前につけることもできる。CHECK 条件は，CHECK 接頭語のリストにある個々の名前ごとに，別々に解禁状態に置かれる。たとえば，接頭語（CHECK（A，B，C））：は（CHECK（A））：（CHECK（B））：（CHECK（C））：と同じ指定である。したがって，各名前ごとに別々の処置を指定することができる。また，名前のリストを与えない場合には，CHECK 接頭語の有効範囲の中にある各変数に対してそれぞれこの条件が解禁状態に置かれる。なお，REVERT ステートメントを使って，リスト内の名前に対して指定されている処置を変更することができる（変更する処置の数に制限はない）。なおまた，NOCHECK 接頭語を使って，特定の名前に対してCHECK 条件を禁止状態に置くことができる。

必要に応じて与える「名前のリスト」は，1つまたはそれ以上の名前をコンマで区切ったものである。この名前は修飾された名前でもよい。この各々の名前は，次のいずれかでなければならない。

1. 入口定数または入口変数。

2. ラベル定数またはラベル変数。

3. 要素，配列，または構造を表わす任意のデータ・タイプで，添字のついていない名前。この変数は i SUB で定義されたものであってはならない。また，この変数は「名前のリスト」の中のロケーターによって明示的に修飾されているものであってはならない。

   データ・リストの中の被定義変数が次のいずれかであってはならない。

   a. 被制御変数に重ねて定義されている変数。

   b. 調節可能な1つ以上の限界をもつ配列に重ねて定義されている変数。

   c. 定数以外のものを指定しているPOSITION 属性をもつものとして定義されている変数。

「名前のリスト」の中の基底付変数には，以下の制約が適用される。

a. この変数は，すでにOFFSET 属性を与えられているものであってはならない。

b. この変数は，REFER オプションによって宣言されている構造のメンバーであってはならない。

c. この変数の基底となっているポインターは，基底付であったり，被定義であってはならないし，パラメーターであってはならない。さらに，このポインターは配列ないし構造のメンバーであってはならない。

CHECK 接頭語で与える名前は，当該ブロックの中で知られている名前またはCHECK 接頭語がつけられているステートメントを指すものである。「名前のリスト」の中で指定することのできる名前の数は，最高２５５である。

CHECK の後に続く「名前のリスト」で構造の名前ないし構造の配列の名前が指定されている場合，このリストは，その構造ないし構造の配列の要素が，宣言されている順に並んでいる名前のリストと同じものである。たとえば，Pが次のように定義されているものとする。

```
DECLARE 1 P，2 Q，2 R，2 S；
```

すると，

```
CHECK （P）
```

は次の指定に等しい。

```
CHECK （P.Q，P.R，P.S）
```

CHECK 接頭語の有効範囲において，次のいずれかの事態が生じたとき，CHECK条件が引き起される。

1. CHECK 接頭語で指定されている名前がステートメント・ラベル定数である場合は，そのラベルが前につけられているステートメントを実行する前にCHECK 条件が提起され，割込みが起こる。ただし，このラベルがFORMAT ステートメントの接頭語である場合には，CHECK 条件は引き起されない。

2. CHECK 接頭語で指定されている名前が（上の一般形式の第３項に示した）変数である場合，このCHECK 接頭語の有効範囲内で，この変数の値またはこの変数の一部分の値が何らかのステートメントによって変更されるたびに，CHECK 条件が引き起される。

とくに，次のような場合にCHECK 条件が引き起される（ここで，識別名ＩＤはこの変数を表わすものとする）。

a． 割当てステートメントの左側にＩＤがある場合。（BY NAME割当てに関しては，言及される名前によってその値が変更される場合に限り，この項の記述が適用される。）

b． 割当てステートメントの左側にある擬似変数の結果としてＩＤがセットされた場合。

c． ＩＤがDOグループの御御変数ないしデータ・リスト内の繰返し指定の制御変数として出て来た場合（または，DOグループの制御変数ないしデータ・リスト内の繰返し指定の制御変数として出て来た擬似変数の結果としてＩＤがセットされた場合）。

d． 編集指示またはリスト指示のGETステートメントのリストの中にＩＤがある場合。

e． データ指示の入力によってＩＤが変更された場合。

f． DISPLAYステートメントのREPLYオプションの中にＩＤがある場合。

g． PUTステートメントのSTRING オプションの中にＩＤがある場合。

h． ＩＤがプログラマーによって定義されたプロシージャーへの引数として渡され，見かけ引数が作成されず，このプロシージャーがRETURNまたはEND で終了した場合。

i． READ ステートメントのKEYTOまたはINTOオプションの中にＩＤがある場合。なお，このREADステートメントがEVENT オプションをもつ場合には，CHECK 条件は引き起されないので注意されたい。

j． ＩＤがロケーター変数としてSETオプションの中にある場合，または，ＩＤが暗示的にセットされた場合。

k． ＩＤがINITIAL 属性によってセットされた非静的変数である場合。

上のa，b，d，e，において，ＩＤがデータ集合体である場合，その集合体の個々の要素に値が与えられるたびにCHECK条件が引き起される点に注意されたい。ＩＤがデータ集合体の１つの要素である場合には，その要素に対してCHECK条件が引き起されるのであり，そのデータ集合体全体に対してこの条件が引き起されるわけではない。

以下に示すいずれの環境においても，CHECK条件は引き起されない。

a． ＩＤまたはＩＤの一部で定義されている変数の値が上述のいずれかの方法で変った場合。

b． 引数ＩＤを表わすパラメーターの値が変った場合。

c． GO TO ないしRETURN ステートメントの中にＩＤがある場合，または，GO TOないしRETURNステートメントの実行を含むステートメントの中にＩＤがある場合。

上に示したいずれの文脈においても，ＩＤは添字のついた形または修飾された形であってよいという点に注意されたい。また，CHECK 接頭語の「名前のリスト」にＩＤがある必要はないという点にも注意されたい。ＩＤは，リストの中の名前に含まれる構造ないし要素を表しているか，リストの中の名前を含む構造ないし要素を表わしていさえすればよい。

ｈの場合を除き，ＩＤへの割当ての直後にCHECK 条件のための割込みが起こる。ｈの場合には，このサブルーチンのRETURNないしENDステートメントを実行した直後にこの割込みが起こる。DOステートメントの場合は，制御がDOステートメントの直後のステートメントに逐次移るたびに割込みが起こる。DOステートメントで実行の繰返しが指定されている場合には，制御変数の値が変るたびに割込みが起こる。

ある１つのステートメントが原因となっていくつかの名前に関連してCHECK 条件が引き起される場合は，リストの中で左にある名前から順に，この条件が起こる。

3. CHECK 接頭語の「名前のリスト」にある識別名が入口定数である場合，この入口定数に対応する入口点の呼出しが行なわれるたびに，この呼出しに先立ってCHECK 条件が提起され，割込みが起こる。ただし，CHECK 条件が提起されるのは，CHECK 接頭語で与えられている入口定数によってこの入口点が呼び出された場合だけである。

結果： CHECK 条件が引き起されても，このとき実行されているステートメントは何の影響も受けない。

システムの標準処置： オン・ユニットが与えられていない場合，「名前のリスト」が与えられているか否かにかかわりなく，次に示すような出力が出される。なお，この出力には現在のステートメント番号も含まれる。

変数または定数	出　　力
算術変数またはストリング変数	名前と値
エリア変数，ファイル変数，入口変数，事象変数，ラベル変数，またはロケーター変数，入口定数またはラベル定数	名前のみ

注：

a. 「名前のリスト」を与えなかった場合，SIGNAL CHECKに対応するシステムの標準処置は，ストリング変数および算術変数の名前（と新しい値）を印刷することだけである。

b. CHECK条件に対するシステムの標準処置では，当該変数をアクセスする必要がある。したがって，割り振られていない変数に対してSIGNAL CHECKが与えられている場合は，エラーが生じる。同様に，オン・ユニットでこの変数にアクセスした場合にもエラーが生じる。

状況： ユーザーの側で特に指定しない限り，CHECKは禁止状態にある。また，NOCHECK条件接頭語の有効範囲の中でもCHECKは禁止状態にある。CHECK接頭語の有効範囲内でだけ，CHECKは解禁状態に置かれる。

正常な戻り： CHECK条件のためのオン・ユニットが正常な形で完了すると，割込みの起こった場所の直ぐ後のところから実行が続行される。

CONDITION（名前）　　プログラマーが名付ける

省略形： COND（名前）

説明： CONDITIONは，該当する名前を指定しているSIGNALステートメントによって引き起される。SIGNALステートメントで指定されている名前によって，どのCONDITION条件を引き起すかが決まる。

ON，SIGNAL，またはREVERTステートメントの中に，ある識別名をもつCONDITIONが記入されているということは，その識別名が文脈の上から宣言されたことを意味する。この識別名にはEXTERNAL属性が与えられる。また，CONDITION属性を使って，識別名を条件名として明示的に宣言することもできる。

システムの標準処置： この条件のためのオン・ユニットが与えられていない場合，システムはメッセージを印刷し，SIGNALの次のステートメントから実行を続行する。

状況： CONDITIONはつねに解禁状態にあり，これを禁止状態にすることはできない。

正常な戻り： オン・ユニットが正常な形で完了すると，この割込みを引き起こしたSIGNALステートメントの次のステートメントから実行が続行される。

CONVERSION　　計算関係

省略形： CONV

説明： 文字ストリング・データに対して有効でない変換が試みられるたびにCONVERSION条件が引き起される。この変換は計算機内部で試みられることもあり，入出力操作の間に試みられることもある。たとえば，ビット・ストリングに変換すべき文字ストリングの中に0または1以外の文字があると，CONVERSION条件が引き起される。また，たとえば，数字フィールドに変換すべき文字ストリングの中にPICTURE指定で許されていない文字があった場合，あるいはコード化算術データに変換すべきストリングの中に算術定数の文字表現が含まれていない場合にもCONVERSION条件が引き起される。

文字ストリング・データはすべて，1文字ずつ左から右に向かって変換される。この間，有効でない文字があるたびにCONVERSION条件が引き起される。有効でない文字が出て来ると，割込みが起こり（もちろん，これはCONVERSIONが禁止状態に置かれていない場合），CONVERSION条件に関して現在指定されている処置がとられる。この処置がオン・ユニットを実行することであるときは，オン・ユニットの中でONSOURCEないしONCHAR擬似変数を使って有効でない文字を訂正することができる。このいずれかの擬似変数を使用している場合は，オン・ユニットから戻った後再び先の変換が試みられる。エラーが訂正されていない場合には，プログラムはループに入ることになる。上述の擬似変数を使用していない場合には，ERROR条件が引き起される。

結果： CONVERSIONが引き起された場合，結果のフィールド全体の内容は，どうなるかわからない。

システムの標準処置： オン・ユニットが与えられていない場合は，システムはメッセージを印刷し，ERROR条件を引き起す。

状況： NOCONVERSIONを指定している条件接頭語の有効範囲を除いて，プログラムを実行している間中CONVERSIONは解禁状態にある。

正常な戻り： この条件のためのオン・ユニットが正常な形で終了した後，当該ストリングの最初のところに制御が戻され，再び変換が試みられる。

ENDFILE（要素ファイル式）　　入 出 力

説明：　ENDFILE条件は，GETまたはREAD操作を行っているときに引き起される可能性がある。すなわち，GETないしREADステートメントで名前が指定されているファイルのファイル区切り符号を越えて読取りを行なおうとすると，この条件が引き起される。この条件が適用されるのは，SEQUENTIAL INPUT，SEQUENTIAL UPDATE，およびSTREAM INPUTのファイルだけである。

レコード方式の入出力においては，READステートメントを実行している間にファイル区切り符号が出て来るたびに，ENDFILE条件が引き起される。

ストリーム方式の入出力においては，あるGETステートメントを実行している間，このGETステートメントのデータ・リストにあるどの項目も転送されないうちに，あるいはあるデータ項目を転送した後他のデータ項目を転送するまでの間にファイル区切り符号が出て来た場合，ENDFILE条件が引き起される。あるデータ項目を転送している間にファイル区切り符号が出て来た場合またはX形式項目を実施している間にファイル区切り符号が出て来た場合には，ERROR条件が引き起される。

ENDFILEが引き起された後当該ファイルが閉じられていない場合には，当該ファイルを対象とするGETないしREADステートメントがこの後出て来るとただちに，またENDFILE条件が引き起される。

EVENTオプションをもつ入出力ステートメントによってENDFILEが引き起された場合には，この事象を対象とするWAITステートメントがこの後実行されるまで割込みは起こらない。

システムの標準処置：　オン・ユニットが与えられていない場合，システムはメッセージを印刷し，ERROR条件を引き起す。

状況：　ENDFILE条件はつねに解禁状態にあり，これを禁止状態に置くことはできない。

正常な戻り：　この条件のためのオン・ユニットが正常な形で終了すると，ENDFILEを引き起したGETないしREADステートメントの直ぐ後のステートメントから実行が続行される（または，EVENTオプションをもつREADによってENDFILEが引き起された場合には，この条件のためのオン・ユニットを呼び出したWAITステートメントに制御が戻される）。

注：　ENDFILE条件のためのオン・ユニットの中でファイルが閉じられた場合には，正常な戻りの後どうなるかはわからない。この種のオン・ユニットでは，GO TOステートメントによって制御をオン・ユニットの外に出さなければならない。

ENDPAGE（要素ファイル式）　　入 出 力

説明：　あるPUTステートメントを実行した結果として，現在のページに対して指定されている限界を越えて新しい行を開始しようとすると，ENDPAGE条件が引き起される。ページの限界は，OPENステートメントのPAGESIZEオプションを用いて指定することができる。PAGESIZEが指定されていない場合には，当コンパイラーの省略時の限界である60行がページの限界であるものとみなされる。データ（編集指示のPUTステートメントの場合にはデータと結び付いている形式項目を含む）を転送している間に，またはLINEオプションまたはSKIPオプションによって，ページの限界を越える新しい行を開始しようとする試みがなされることがある。また，現在の行番号より小さい行番号を指定しているLINEオプションまたはLINE形式項目によって，ENDPAGEが引き起されることもある。

ENDPAGEが引き起された時点の行番号は，PAGESIZEオプションで指定されている番号より1つ大きい（または，PAGESIZEが指定されていない場合には，61）。したがって，同じページに印刷を続けることも可能である。オン・ユニットでは，現在の行番号を1にセットするためのPAGEオプションまたはPAGE形式項目を実行することによって，新しいページを開始することができる。

ENDPAGEは，1つのページでは1回しか引き起されない。このため，オン・ユニットで新しいページが開始されなければ，現在の行番号が無限に大きくなることがある。この後出て来たLINEオプションまたはLINE形式項目で現在の行番号より小さい行番号が指定されていた場合には，ENDPAGEは引き起されないが現在の行番号が1にセットされて，新しいページが開始される。ただし，指定されている行番号が現在の行番号に等しく，しかもファイルの位置がこの行の1桁目にある場合にはENDPAGEは引き起されない。

データを転送している間にENDPAGEが引き起された場合には，オン・ユニットから戻った後，データが現在の行に印刷される（ここでいう現在の行は，オン・ユニットで変更されている可能性がある）。LINEないしSKIPオプションによってENDPAGEが引き起された場合には，オン・ユニットから戻った後，そのLINEないしSKIPで指定されている処置は無視される。

システムの標準処置：　オン・ユニットが与えられていない場合，システムは新しいページを開始する。SIGNALステートメントによってこの条件が引き起された場合には，実行はこれによって何の影響も受けず，SIGNALステートメントの次のステートメントから実行が続行される。

状況：　ENDPAGEはつねに解禁状態にあり，これを禁止状態に置くことはできない。

正常な戻り：　この条件のためのオン・ユニットが正常な形で完了すると，上述のようにして当該PUTステートメントの実行が続行される。

ERROR　　システムの処置

説明：　次のような環境においてERROR条件が引き起される。

1. システムの標準処置が「エラー・メッセージを印刷しERROR条件を引き起す」ことであるようなON条件に

対するシステムの標準処置の結果として。

2. プログラムを実行している間に発生した（ON条件のない）エラーの結果として。

3. SIGNAL ERRORステートメントの結果として。

システムの標準処置： FINISH 条件が引き起され，その後当該プログラムが停止される。

状況： ERROR はつねに解禁状態にあり，これを禁止状態に置くことはできない。

正常な戻り： オン・ユニットが正常な形で完了すると，システムの標準処置がとられる。

FINISH　　　システムの処置

説明： FINISH 条件は，PL／Iプログラムを停止させるようなステートメント（STOP ステートメント，または，第1プロシージャーの中のRETURNないしENDステートメント）を実行している間に引き起される。また，SIGNAL FINISH によってこの条件が引き起されることもあるし，ERROR条件のためのシステムの標準処置の一部としてこの条件が引き起されることもある。

システムの標準処置： オン・ユニットが与えられていない場合は，何の処置もとられない。すなわち，割込みを引き起こしたステートメントの実行が続行される。

状況： FINISH はつねに解禁状態にあり，これを禁止状態に置くことはできない。

正常な戻り： オン・ユニットが正常な形で完了すると，割込みを引き起こしたステートメントの実行が続行される。

FIXEDOVERFLOW　　　計　算

省略形： FOFL

説明： 固定小数点数算術演算の結果の長さが当コンパイラーにおいて許されている最大の長さを越えたとき，FIXEDOVERFLOW条件が引き起される。この最大の長さは，10進固定小数点数値の場合には15であり，2進固定小数点数値の場合には31である。

結果： 有効でない固定小数点数演算の結果は，定義できない。

システムの標準処置： オン・ユニットが与えられていない場合，システムはメッセージを印刷し，ERROR 条件を引き起す。

状況： NOFIXEDOVERFLOWと指定されている条件接頭語の有効範囲を除いて，プログラムを実行している間中FIXEDOVERFLOWは解禁状態にある。

正常な戻り： この条件のためのオン・ユニットが正常な形で終了すると，割込みが起こった場所の直後に制御が戻される。

KEY（要素ファイル式）　　　入　出　力

説明： KEY条件は，キー付レコードを対象とする操作を行なっているときにだけ，引き起される可能性がある。次にあげるいずれかの事態が生じたときにこの条件が引き起される。

1. 該当のキー付レコードが見つからない。

2. 重複したキーを追加しようとした。（編成方式がINDEXED，キー順のVSAM，またはREGIONAL(1)のファイルのみ）

3. キーの順序が間違っている。

4. キーを変換している間にエラーが発生した。

5. キーがナル・ストリングであるか，見かけのレコード・ストリング(8)'1'Bで始まっている。

6. キー付レコードを追加するためのスペースが使用可能でない。

7. キーが当該データ・セットの限界の外にある（REGIONALデータ・セットのみ）。

8. INDEXED データ・セットまたはキー順のVSAM データ・セットを対象とする順次方式のREWRITE〔FROM〕操作の間に，内蔵されたキーを変更しようとした。

9. キーが当該データ・セットで定義されているキー範囲の外にある（キー順VSAMのみ）。

EVENT オプションを使用している入出力ステートメントによってKEYが引き起された場合は，この後同じプロシージャーの中でこの事象を対象とするWAITステートメントが実行されるまで，割込みは起こらない。

REGIONAL(3) データ・セットを対象とする操作においてLOCATEステートメントが使用されており，しかも指定されている区域にあいているスペースが充分にない場合，レコードの転送が試みられるまではKEY条件は引き起されない。同様に，キー順のVSAMデータ・セットを対象とする操作においてLOCATEステートメントが使用されており，キーがキーの定義域の外にあり，しかも指定されている区域にあいているスペースが充分にない場合，レコードの転送が試みられるまではKEY条件は引き起されない。

この条件を引き起こす原因となったレコードも現在のレコードも転送されない。

システムの標準処置：　オン・ユニットが与えられていない場合，システムはメッセージを印刷し，ERROR 条件を引き起す。

状況：　KEYはつねに解禁状態にあり，これを禁止状態に置くことはできない。

正常な戻り：　この条件のためのオン・ユニットが正常な形で完了すると，KEY条件を引き起す原因となったステートメントの直ぐ後のステートメントに制御が渡される（または，EVENTオプションをもつ入出力ステートメントによってKEYが引き起された場合には，KEY条件のためのオン・ユニットを呼び出したWAITステートメントに制御が戻される）。

注：　この条件のためのオン・ユニットにおいてファイルが閉じられた場合，正常な戻りの後どのような処置がとられるかは定義できない。このようなオン・ユニットの場合は，GOTOステートメントによってオン・ユニットの外に制御を渡すべきである。

NAME（要素ファイル式）　入出力

説明：　NAME条件は，FILEオプションをもつデータ指示方式のGETステートメントを実行しているときにだけ，引き起される可能性がある。この条件は，次の場合のように，認定のできない要素変数がストリームの中に出て来た場合に引き起される。

1. 変数の中に有効でない文字がある場合：

   変数の中にブランク以外の区切り符号（コンマ，セミコロン，またはファイル終りマーク）がある。

   右括弧と等号の間にブランク以外の文字がある。

   数字でない添字が付いている。

2. 変数の中に有効でないブランクがある場合：

   名前の中または添字の値の中にブランクがある（注：　修飾された名前に関してはそのピリオドのいずれの側にブランクがあってもよいし，また，添字の符号と数字の間にブランクがあってもよい）。

3. 名前がない場合または名前が有効でない場合：

   データ・リストの中に対応する名前がない。

   データ・リストがないときに，当該ブロックにおいて名前が知られていない。

   修飾された名前が完全に修飾されてはいない。

   完全に修飾された名前が２５６文字を越えている。

   iSUBで定義された名前である。

4. 添字のリストがない場合または添字のリストが有効でない場合：

   添字が抜けている。

   添字の数が間違っている。

   添字が６桁以上の数である（先行ゼロは無視される）。

   添字の値が範囲の外にある。

プログラマーは，オン・ユニットの中でDATAFIELD 組込み関数を用いて，当該データ・フィールドを取り出すことができる。

システムの標準処置：　オン・ユニットが与えられていない場合，システムは，間違いのあるデータ・フィールドを無視し，メッセージを印刷し，当該GETステートメントの実行を続ける。

状況：　NAMEはつねに解禁状態にあり，これを禁止状態に置くことはできない。

正常な戻り：　この条件のためのオン・ユニットが正常な形で完了すると，ストリームの中の次の識別名から当該GETステートメントの実行が続行される。

OVERFLOW　計　算

省略形：　OFL

説明：　OVERFLOW 条件は，浮動小数点数の大きさが許容最大値を越えたときに引き起される。浮動小数点数またはその中間結果の大きさは，約10^{75}，すなわち 2^{252} を越えてはならない。

結果：　有効でない，このような浮動小数点数の値は定義できない。

システムの標準処置：　オン・ユニットが与えられていない場合，システムはメッセージを印刷し，ERROR条件を引き起す。

状況：　NOOVERFLOW が指定されている条件接頭語の有効範囲を除いて，プログラムを実行している間中OVERFLOW はいつも解禁状態にある。

正常な戻り：　この条件のためのオン・ユニットが正常な形で終了すると，割込みが起こった場所の直ぐ後に制御が戻される。

RECORD（要素ファイル式）　入出力

説明：　RECORD条件はREAD, WRITE, LOCATE,またはREWRITE 操作を行なっているときにだけ引き起される可能性がある。次のいずれかの事態が生ずると，この条件が引き起される。

1. 固定長レコード用ファイルに対して指定されているレコード長がREAD　INTOステートメントの変数より小さく，しかもこの変数が固定長ストリングのとき。この変数の残りの部分は，どうなるかわからない。

2. レコードがREAD　INTOステートメントの変数より大きいとき。レコードの残りの部分は失われる。

3. 最大レコード長がWRITE, REWRITE, またはLOCATEステートメントの変数より小さいとき。この場合，WRITEまたはREWRITEに関しては，変数の残りの部分が失われる。LOCATEに関しては，変数は転送されない。

4. 固定長レコード用ファイルに対して指定されているレコード長がWRITE，REWRITE，またはLOCATEステートメントの変数より大きい。この場合，レコードの残りの部分はどうなるかわからない。この変数が可変長ストリングである場合，SCALARVARYING オプションがこのファイルに適用されているならば，RECORD は引き起ざれない。

5. WRITEまたはREWRITEステートメントの変数の長さがゼロであると指示されているとき。転送は行なわれない。この変数が可変長ストリングである場合，SCALARVARYINGオプションがこのファイルに適用されているならば，RECORDは引き起されない。

6. WRITEまたはREWRITEステートメントの変数が短すぎて，当該データ・セットの内蔵されたキーがこの変数に入らないとき。この場合，転送は行なわれない。（キー順VSAMのみ）。VSAMファイルにおいては，さらにRECORD条件が検知されるという点を除き，レコードのチェックは，可変長レコードと同様にして行なわれる。

エリアを転送している間にRECORD 条件が引き起された場合，誤った情報がこのエリア制御フィールドに入っていることがある。

SCALARVARYING オプションが当該ファイルに適用されている場合（可変長ストリングを転送するには，位置指定方式でこのオプションをファイルに適用しなければならない），要素可変長ストリングと共に2バイトの長さ接頭語が転送される。RECORD条件が引き起されても，この長さ接頭語はリセットされない。SCALARVARYINGオプションが当該ファイルに適用されていない場合，長さ接頭語は転送されず，入力の際レコード長とストリングの最大長のうち短い方が，可変長ストリングの現在の長さになる。

EVENTオプションを使用している入出力ステートメントによってRECORD条件が引き起された場合，この後同じプロシージャーの中でその事象を対象とするWAIT ステートメントが実行されるまで，割込みは起こらない。

LOCATEステートメントでRECORD条件が引き起された場合，そのポインター変数はセットされず，次の入出力ステートメントを実行するときにも緩衝域の中のデータは転送されない。

CONSECUTIVEデータ・セットからSEQUENTIAL UNBUFFEREDファイルに読み取られた不定長レコードに対して，RECORD条件が引き起されることはない。

システムの標準処置：　オン・ユニットが与えられていない場合，システムはメッセージを印刷し，ERROR条件を引き起す。

状況：　RECORD はつねに解禁状態にあり，これを禁止状態に置くことはできない。

正常な戻り：　オン・ユニットが正常な状態で完了すると，RECORD条件を引き起したステートメントの直後のステートメントから実行が続行される（または，EVENTオプションをもつ入出力ステートメントによってRECORDが引き起された場合には，オン・ユニットを呼び出したWAITステートメントに制御が戻る）。

注：　この条件のためのオン・ユニットの中でファイルが閉じられた場合，正常な戻りの後どのような処置がとられるかはわからない。この種のオン・ユニットでは，GO TO ステートメントによってオン・ユニットの外に制御を渡すべきである。

SIZE　計算

説明：　変数ないし中間結果への割当ての際，または入出力操作を行なっている間に2進数ないし10進数の最高位の（左端の）有効数字が失われた場合にだけ，SIZE 条件が引き起される。異なるデータ・タイプ，異なる基数，異なるスケール，または異なる精度が関係する変換が行なわれるときに，このように最高位の有効数字が失われることがある。

SIZE条件とFIXEDOVERFLOW条件との相違は，計算された固定小数点数値の大きさが当コンパイラーにおいて許されている最大値を越えたときに，FIXEDOVERFLOW 条件が引き起されるのに対して，SIZE条件の場合は，あるデータ項目に割り当てられる値の大きさがそのデータ項目について宣言されている（または省略時の）大きさを越えたときに引き起されるという点である（FIXEDOVERFLOW条件に関しては，この条件の項を参照されたい）。ある値を割り当てる際にSIZE条件が引き起されるかどうかは，その値を計算したときにFIXEDOVERFLOW条件が起ったかどうか，ということとは無関係である。

あるデータ項目について宣言されている大きさは，必ずしもその項目が記憶域の中に確保している精度ではない。しかし，SIZE条件が引き起される基準となる限界は，宣言されている大きさまたは省略時の大きさであって，記憶域における実際の大きさではない。たとえば，精度が（20）の固定小数点2進数項目は，記憶域の中では1フルワードを占めるが，FIXED BINARY(20)を越える大きさの値がこの項目に割り当てられると，SIZE条件が引き起される。

結果： 大きさが適当でない値の受取り側データ項目の内容は，どうなるかわからない。

システムの標準処置： オン・ユニットが与えられていない場合，システムはメッセージを印刷し，ERROR条件を引き起す。

状況： SIZEが指定されている条件接頭語の有効範囲を除いて，プログラムの実行中は，NOSIZE条件接頭語の有効範囲およびその他の範囲においても，SIZEは禁止状態にある。

正常な戻り： この条件のためのオン・ユニットが正常な形で終了すると，割込みの発生した場所の直後に制御が戻される。

STRINGRANGE　　プログラム・チェックアウト

省略形： STRG

説明： SUBSTR参照の引数の長さがSUBSTR組込み関数の項に記述されている規則に合致しないことがあると，そのたびにSTRINGRANGE条件が引き起される。この条件は，このようなSUBSTR参照があるたびに引き起される。

システムの標準処置： メッセージが印刷され，「正常な戻り」の項に記述されている処理が続行される。

状況： ユーザーの側で特に指定しない限り，STRINGRANGEは禁止状態にある。また，NOSTRINGRANGE条件接頭語の有効範囲においてもこの条件は禁止状態にある。STRINGRANGE条件接頭語の有効範囲の中でだけ，この条件は解禁状態に置かれる。

正常な戻り： オン・ユニットからの正常な戻りの後，訂正されたSUBSTR参照の実行が続行される。このSUBSTR参照の値は，次のように定義される。

（オン・ユニットが与えられている場合には，その実行の後の）原ストリングの長さがk，開始点がi，サブストリングの長さがjであるとすれば，

1. iがkより大きい場合，値はナル・ストリングである。
2. iがkに等しいかkより小さい場合，値は，原ストリングのm番目の文字またはビットから始まる，n個の文字またはビットを含むサブストリングである。ここで，mとnは次のように定義される。

m=MAX(i,1)

n=MAX(0,MIN(j+MIN(i,1)-1,
　k-m+1))
　〔jが指定されている場合〕

n=k-m+1
　〔jが指定されていない場合〕

以上のようにして，新しい引数は，強いて限界内に収められるわけである。

iとjの値は，オン・ユニットに入る前に計算されるだけであり，オン・ユニットから戻った後これらの値が再度計算されることはない。

SUBSTRの最初の引数が長さの変わるストリングである場合には，オン・ユニットの中でkの値が変化することがある。値nは，オン・ユニットから戻った後kの新しい値を使って計算される。

STRINGSIZE　　プログラム・チェックアウト

省略形： STRZ

説明： STRINGSIZE条件は，あるストリングがそれより短いストリングに割り当てられたときに引き起される。この場合，目標ストリングに原ストリングを収容することができるように，原ストリングの右端の方の文字ないしビットが切り捨てられる。

結果： 切捨てが行なわれた後のストリングが，その目標ストリングに割り当てられる。

システムの標準処置： メッセージが印刷され，処理が続行される。

状況： ユーザーの側で特に指定しない限り，STRINGSIZEは禁止状態にある。また，NOSTRINGSIZE条件接頭語の有効範囲においてもこの条件は禁止状態にある。STRINGSIZE条件接頭語の有効範囲の中でだけ，この条件は解禁状態に置かれる。

正常な戻り： オン・ユニットからの正常な戻りの後，この割当ての直後の場所から実行が続行される。

SUBSCRIPTRANGE　　プログラム・チェックアウト

省略形： SUBRG

説明： 添字の値を計算した結果，その値がこの添字の指定されている限界の外にあることがわかると，SUBSCRIPTRANGEが引

き起される可能性がある。また，iSUB の添字が iSUB 定義の配列の宣言で与えられている範囲の外にあるときにも，この条件が引き起される。他の添字の評価との間で相対的にみて，SUBSCRIPTRANGEが引き起される順序は，定義できない。

結果：　SUBSCRIPTRANGEが引き起された場合，有効でない添字の値は定義できない，したがって，その参照の結果がどうなるかもわからない。

システムの標準処置：　オン・ユニットが与えられていない場合，システムはメッセージを印刷し，ERROR 条件を引き起す。

状況：　SUBSCRIPTRANGEは，ユーザーの側で何も指定しないと禁止状態になる。また，NOSUBSCRIPTRANGE 条件接頭語有効範囲においても，この条件は禁止状態にある。SUBSCRIPTRANGE条件接頭語の有効範囲においてのみ，この条件は解禁状態に置かれる。

正常な戻り：　SUBSCRIPTRANGEオン・ユニットからの正常な戻りの後，ERROR条件が引き起される。

TRANSMIT（要素ファイル式）　入出力

説明：　入出力操作を行なっているときにはそれがどのようなタイプの入出力操作であっても，TRANSMIT 条件が引き起される可能性がある。この条件は，永続的な転送エラーによって引き起される。したがって，これは転送されたデータが誤っている可能性のあることを表わしている。

入力の際には，誤りがある可能性のあるレコードを割り当てた後でTRANSMIT 条件が引き起される。レコードがブロック化されている場合，このブロックの中のこれより後にあるレコード各々についてTRANSMIT条件が引き起される。出力の際には，誤りがある可能性のあるデータ項目を転送した後でTRANSMIT 条件が引き起される。レコードがブロック化されている場合は，ブロックが一杯になったときに転送が行なわれる。出力ステートメントの後で各々転送が行なわれるわけではない。

EVENTオプションを使用している入出力ステートメントによってTRANSMIT が引き起された場合は，この後同じプロシージャーの中でその事象を対象とするWAITステートメントが実行されるまで，割込みは起らない。

システムの標準処置：　オン・ユニットが与えられていない場合，システムはメッセージを印刷し，ERROR条件を引き起す。

状況：　TRANSMIT はつねに解禁状態にあり，これを禁止状態に置くことはできない。

正常な戻り：　オン・ユニットが正常な状態で完了すると，エラーが発生しなかったかのように処理が続行され，TRANSMIT 条件を引き起す原因となったデータ項目のステートメントによって他の条件（たとえば，RECORD）を引き起すことも可能になる。（EVENTオプションをもつ入出力ステートメントによってTRANSMIT が引き起された場合には，オン・ユニットを呼び出したWAITステートメントに制御が戻される。）

注：　この条件のためのオン・ユニットの中でファイルが閉じられた場合，正常な戻りの後どのような処置がとられるかは定義できない。この種のオン・ユニットの中では，オン・ユニットの外に制御を渡すときは，GO TOステートメントを使用すべきである。

UNDEFINEDFILE（要素ファイル式）　入出力

省略形：　UNDF（要素ファイル式）

説明：　ファイルを開こうとして失敗したときはつねに，UNDEFINEDFILE 条件が引き起される。2つ以上のファイル名を指定しているOPEN ステートメントによって開こうとした場合は，そのステートメントで指定されている他のファイルを開こうとする試みがすべて行なわれた後でUNDEFINEDFILE 条件が引き起される。同一のOPENステートメントにおいて2つ以上のファイルに関してこの条件が引き起された場合は，そのOPEN ステートメントの中でファイル名が並んでいる順に（左から右に向かって）オン・ユニットが実行される。

EVENTオプションをもたない入出力ステートメントの中で暗示的にファイルを開くことによって，この条件が引き起された場合は，オン・ユニットから正常に戻ったあとで，割込みを引き起こした入出力ステートメントの残りの部分の処理が続行される。オン・ユニットにおいてこのファイルが開かれなかった場合には，この入出力ステートメントの実行を続けることができず，このためERROR条件が引き起される。

EVENT オプションをもつ入出力ステートメントの中で暗示的にファイルを開くことによって，この条件が引き起された場合は，この事象変数が初期設定される前に割込みが起こる。いいかえれば，この事象変数は前の値を保持しており，非活用状態にとどまっている。オン・ユニットから正常に戻った後，この事象変数が初期設定される。つまり，この変数は活用状態になり，その完了値が '0'Bにセットされる。ただし，これはオン・ユニットの中でファイルが開かれた場合である。この後，割り込まれたステートメントの残りの部分の処理が続行される。オン・ユニットの中でファイルが開かれなかった場合には，事象変数の初期設定は行なわれないので，ステートメントの実行を続けることができない。このため，この場合にはERROR条件が引き起される。

このコンパイラーでは，たとえば，次のような場合にUNDEFINEDFILE 条件が引き起される。

1. ENVIRONMENTオプションのいくつかが矛盾しているか，誤ったオプションがある。

2. VSAMデータ・セットを開くときにVSAMによってエラーが検出された。

システムの標準処置：　オン・ユニットが与えられていない場合，システムはメッセージを印刷し，ERROR条件を引き起す。

状況：　UNDEFINEDFILEはつねに解禁状態にあり，これを禁止状態に置くことはできない。

正常な戻り：　最後のオン・ユニットが正常な状態で完了した後，この条件を引き起こす原因となったステートメントの直後のステートメントに制御が与えられる（暗示的な開放の場合にとられる処置については，「説明」の項を参照のこと）。

UNDERFLOW　計　算

省略形：　UFL

説明：　浮動小数点数の大きさが許容最小値より小さいと，UNDERFLOW条件が引き起される。（システム／360およびシステム／370においては，ゼロでない浮動小数点数値は，約10^{-78}，すなわち2^{-260}より小さいものであってはならない。）

同じ数を減算してもUNDERFLOWは引き起されない（これは，有効数字エラーと呼ばれることが多い）。

式X＊＊（－Y）　（ここで，Y＞0）の値を計算するときには，X＊＊Yの逆数をとることによって計算することができる点に注意されたい。このため，UNDERFLOW条件ではなく，OVERFLOW条件が引き起されることがある。

結果：　有効でない浮動小数点数値がゼロにセットされる。

システムの標準処置：　オン・ユニットが与えられていない場合，システムはメッセージを印刷し，割込みが発生した場所から実行を続行する。

状況：　NOUNDERFLOWが指定されている条件接頭語の有効範囲を除いて，プログラムを実行している間中，UNDERFLOWは解禁状態にある。

正常な戻り：　この条件のためのオン・ユニットが正常な形で終了すると，割込みが発生した場所の直後に制御が戻される。

ZERODIVIDE　計　算

省略形：　ZDIV

説明：　ゼロで割ろうとすると，ZERODIVIDE条件が引き起される。固定小数点数除算の場合にも，浮動小数点数除算の場合にも，この条件が引き起される。

結果：　ゼロによる除算の結果はわからない。

システムの標準処置：　オン・ユニットが与えられていない場合，システムはメッセージを印刷し，ERROR条件を引き起す。

状況：　NOZERODIVIDEが指定されている条件接頭語の有効範囲を除いて，プログラムを実行している間中，ZERODIVIDEは解禁状態にある。

正常な戻り：　この条件のためのオン・ユニットが正常な形で終了すると，割込みが発生した場所の直後に制御が戻される。

I節：　属　　性

本節は，すべての属性についての詳細な説明をアルファベット順に収録したものである。択一的属性は一個所にまとめて説明する。

図I．1は属性に関する個々の規則をもとにしてまとめたものであり，以下の事柄の「早見表」として使えるよう工夫してある。

1. 属性のデータ・タイプ別分類。

2. ある1つのデータ項目に適用することができる属性の有効な組合せ。

   変数に関しては，データ属性，有効範囲属性，記憶域属性，境界合せ属性の欄から属性を選び出さなければならない。表に示されているタイプの定数に関しては，データ属性および有効範囲属性の欄から属性を選び出さなければならない。ある1つのデータ項目に適用される全属性は，明示宣言または文脈宣言，およびプログラマーの定義になる省略時属性または標準省略時属性から得られることに注意していただきたい。

3. 互いに両立しない属性。

   ある1つのデータ・タイプに適用されるものとして示されている属性は，他のデータ・タイプに適用される属性とは両立しない。ただし，2つのデータ・タイプに適用されるものとして示されている属性はこの限りでない。ある1つのデータ・タイプの中での択一的属性（たとえば，BIT とCHARACTER）も両立しない。

例を用いてこの図の使い方を説明しよう。

```
DECLARE ST BIT(10);
```

このような宣言が与えられていると，STという名前には，標準省略時属性であるAUTOMATIC，INTERNAL，UNALIGNEDが適用される。

以下の注は，図I．1の注1，注2，注3に対応するものである。

1. 暗示的に宣言されている識別名（または，有効範囲属性，記憶域属性，ないし境界合せ属性だけが明示的に宣言されている識別名）は，算術変数であるとみなされる。この識別名の頭文字がIからNの間の文字である場合，標準省略時属性は，FIXED BINARY （15，0）である。これ以外のすべての場合には，標準省略時属性はFLOAT DECIMAL(6)である。
   BINARY，DECIMAL，REAL，またはCOMPLEXと指定されている場合には，標準省略時属性はFLOATである。このほか，スケール因数によって精度が指定されている場合には，標準省略時属性はFIXEDである。

2. IRREDUCIBLE，REDUCIBLE，RETURNS,または OPTIONSは，ENTRYを暗示している。なお，入口定数がパラメーター属性をもつことがある。

3. ファイル属性と，ENVIRONMENT属性のオプションとの関係は，図I．2に示されている。なお，ファイル定数がパラメーター属性をもつこともできる。

図I．2は，図I．1のファイル定数関係の記入事項を拡大したものであり，データ・セットの編成タイプごとにファイル属性とENVIRONMENT属性のオプションとの関係が示してある。また，この図には，各々のファイル属性がどのような属性を暗示しているかということも示されている。

MEDIUMオプションについては，「第10章：　入力と出力」で説明されている。これ以外のオプションについては，「第11章：　ストリーム方式の転送」および「第12章：　レコード方式転送」で説明されている。

<table>
<tr><th>データ・タイプ</th><th>データ属性</th><th>有効範囲属性</th><th>記憶域属性</th><th>境界合せ属性</th></tr>
<tr><td>算術変数 [1]</td><td>REAL｜COMPLEX
FLOAT｜FIXED
BINARY｜DECIMAL
（精 度）</td><td rowspan="10">{INTERNAL}
{EXTERNAL}

AUTOMATIC，
BASED，
DEFINED，
パラメーター
に関しては標準省略時属性は
INTERNAL であり，必須のものである。
CONTROLLED，
STATIC
に関しては標準省略時属性はINTERNALである。</td><td rowspan="10">記憶域クラス：
AUTOMATIC
STATIC
BASED
CONTROLLED

INTERNALの場合にはAUTOMATICが，EXTERNALの場合にはSTATICが，それぞれ標準省略時属性である。

〔INITIAL〕

定義：
DEFINED
〔POSITION〕
単純パラメーター：
パラメーター
〔CONNECTED〕
被制御パラメーター：
パラメーター
CONTROLLED
〔INITIAL〕</td><td>{ALIGNED}
{UNALIGNED}</td></tr>
<tr><td>ストリング変数</td><td>BIT｜CHARACTER
〔（長 さ）〕
〔VARYING〕</td><td rowspan="2">{ALIGNED}
{UNALIGNED}</td></tr>
<tr><td>ピクチャー変数</td><td>{PICTURE}
{REAL｜COMPLEX-PICTURE}</td></tr>
<tr><td>ラベル変数</td><td>LABEL</td><td rowspan="4">{ALIGNED}
{UNALIGNED}</td></tr>
<tr><td>ファイル変数</td><td>FILE〔VARIABLE〕</td></tr>
<tr><td>入口変数 [2]</td><td>ENTRY
〔RETURNS〕〔OPTIONS〕
〔VARIABLE〕</td></tr>
<tr><td>ロケーター変数</td><td>POINTER｜{OFFSET〔（エリア変数）〕}</td></tr>
<tr><td>エリア変数</td><td>AREA〔（サイズ）〕</td><td rowspan="3">ALIGNED</td></tr>
<tr><td>事象変数</td><td>EVENT</td></tr>
<tr><td>タスク変数</td><td>TASK</td></tr>
</table>

<table>
<tr><td>ファイル定数 [3]</td><td>FILE ENVIRONMENT
STREAM｜RECORD
INPUT｜OUTPUT｜UPDATE
SEQUENTIAL｜DIRECT｜
TRANSIENT
BUFFERED｜UNBUFFERED
KEYED BACKWARDS
PRINT EXCLUSIVE</td><td>{INTERNAL}
{EXTERNAL}</td></tr>
<tr><td>入口定数</td><td>（VARIABLEを除き，入口変数と同じ）</td><td>EXTERNAL</td></tr>
<tr><td>組込み入口定数</td><td>BUILTIN</td><td rowspan="2">INTERNAL</td></tr>
<tr><td>総称入口定数</td><td>GENERIC</td></tr>
<tr><td>条件定数</td><td>CONDITION</td><td>{INTERNAL}
{EXTERNAL}</td></tr>
</table>

集合体変数

配列：　変数の宣言に（次元）を追加することができる。

構造：　構造の中の名前に対して指定できる属性は，その名前が宣言されているレベルによって異なる。

1. 大構造名の場合には，データ・タイプを与えることができない。
   LIKE属性は，指定することができる。

2. 小構造名の場合には，データ・タイプ，有効範囲属性，および記憶域属性を与えることができない。
   LIKE属性は指定することができる。

3. 要素名の場合には，有効範囲属性および記憶域属性を与えることができない。

下線は標準省略時属性を示している。注については，前ページを参照のこと。

図I.1　属性のデータ・タイプ別分類

データ・セット編成のタイプ / 適用される属性	STREAM	RECORD SEQUENTIAL BUF CONSECUTIVE	RECORD SEQUENTIAL BUF INDEXED	RECORD SEQUENTIAL BUF REGIONAL	RECORD SEQUENTIAL UNBUF CONSECUTIVE	RECORD SEQUENTIAL UNBUF REGIONAL	RECORD DIRECT INDEXED	RECORD DIRECT REGIONAL	暗示される属性
FILE ATTRIBUTES: BACKWARDS	–	O	–	–	O	–	–	–	FILE RECORD SEQUENTIAL INPUT
BUFFERED	–	D	D	D	–	–	–	–	FILE RECORD SEQUENTIAL
DIRECT	–	–	–	–	–	–	S	S	FILE RECORD KEYED
ENVIRONMENT	I	S	S	S	S	S	S	S	FILE
FILE	I	I	I	I	I	I	I	I	
INPUT	D	D	D	D	D	D	D	D	FILE
KEYED	–	–	O	O	–	O	I	I	FILE RECORD
OUTPUT	O	O	O	O	O	O	–	O	FILE
PRINT	O	–	–	–	–	–	–	–	FILE STREAM OUTPUT
RECORD	–	I	I	I	I	I	I	I	FILE
SEQUENTIAL	–	D	D	D	D	D	–	–	FILE RECORD
STREAM	D	–	–	–	–	–	–	–	FILE
UNBUFFERED	–	–	–	–	S	S	–	–	FILE RECORD SEQUENTIAL
UPDATE	–	O	O	O	O	O	O	O	FILE RECORD

凡例:

I　属性を指定するか，暗示しなければならない。
D　省略時属性
O　必要な場合にのみ指定する任意選択属性。
S　属性を指定しなければならない。
–　有効でない属性。

注:

1. テープ・ファイルの場合にはUPDATEは有効でない。
2. BACKWARDSはテープ・ファイルの場合にのみ有効である。
3. INDEXED出力およびREGIONAL出力の場合には，KEYEDを指定しなければならない。

図I.2　ファイル宣言（2の1）

データ・セット編成のタイプ / 該当するオプション	STREAM	RECORD SEQUENTIAL BUF CONSECUTIVE	RECORD SEQUENTIAL BUF INDEXED	RECORD SEQUENTIAL BUF REGIONAL	RECORD SEQUENTIAL UNBUF CONSECUTIVE	RECORD SEQUENTIAL UNBUF REGIONAL	RECORD DIRECT INDEXED	RECORD DIRECT REGIONAL
ENVIRONMENT オプション: ADDBUFF(n)	–	–	–	–	–	–	O	–
ASCII	O	O	–	–	–	–	–	–
BLKSIZE(n)	I	I	I	I	I	I	I	I
BUFFERS({1|2})	I	I	I	I	–	–	–	–
BUFOFF[(n)]	O	O	–	–	–	–	–	–
COBOL	O	O	O	O	O	O	O	O
CONSECUTIVE	–	D	–	–	–	–	–	–
CTLASA|CTL360	–	O	–	–	O	–	–	–
EXTENTNUMBER(n)	–	–	I	I	–	I	I	I
F|FB	–	–	S	–	–	–	S	–
F|FB|D|DB|U	S	S	–	–	–	–	–	–
F|FB|V|VB|U	I	S	–	–	S	–	–	–
F|U	–	–	–	S	–	S	–	S
FUNCTION	O	O	–	–	–	–	–	–
GENKEY	–	–	O	–	–	–	–	–
HIGHINDEX(n)	–	–	O	–	–	–	O	–
INDEXAREA(n)	–	–	O	–	–	–	O	–
INDEXED	–	–	S	–	–	–	S	–
INDEXMULTIPLE	–	–	O	–	–	–	O	–
KEYLENGTH(n)	–	–	S	S	–	S	S	S
KEYLOC(n)	–	–	O	–	–	–	O	–
LEAVE	O	O	–	–	O	–	–	–
MEDIUM	I	S	S	S	S	S	S	S
NOLABEL	O	O	–	–	O	–	–	–
NOTAPEMK	O	O	–	–	–	–	–	–
NOWRITE	–	–	–	–	–	–	O	–
OFLTRACKS(n)	–	–	O	–	–	–	–	–
RECSIZE(n)	I	I	I	I	I	I	I	I
REGIONAL({1,3})	–	–	–	S	–	S	–	S
SCALARVARYING	O	O	O	O	O	O	O	O
UNLOAD	O	O	–	–	O	–	–	–
VERIFY	O	O	O	O	O	O	O	O

凡例：

I オプションを指定するか暗示するかしなければならない。

D 省略時オプション。

O 任意選択のオプション：必要な場合にのみ指定する。

S オプションを指定しなければならない。

– 有効でないオプション。

注：

1. CONSECUTIVE，INDEXED，およびREGIONALのファイルに対しては，BLKSIZEまたはRECSIZEのどちらか，またはこの双方を指定しなければならない。
2. F,FB,D,DB,UはASCIIデータ・セットに対してのみ有効である。
3. CTLASA/CTL360はASCIIデータ・セットに対しては有効でない。
4. KEYLENGTH(n)はREGIONAL(1)ファイルに対しては有効でない。
5. REGIONAL(1)ファイルに対してはFだけが有効である。
6. NOWRITE は UPDATE ファイルに対してのみ有効である。
7. GENKEY は INPUT またはUPDATEファイルに対してのみ有効である。この場合，KEYEDが必要である。

図I.2 ファイル宣言（2の2）

ALIGNED と UNALIGNED

省略形：　UNALIGNEDに対してはUNAL

ALIGNED 属性とUNALIGNED 属性とは，それぞれアクセス速度を上げたり，記憶域を節約したりする目的で，記憶域におけるデータ要素の位置設定を指定するためのものである。要素変数，配列変数，または構造変数にこれらの属性を与えることができる。

ALIGNEDは，データ要素をそのデータ・タイプの要件に対応する記憶域境界に揃えるよう指定するためのものである。

UNALIGNEDは，ビット・ストリングの場合は，次に使用可能なビット境界にマップし，また，ハーフワード項目，ワード項目，ないしダブルワード項目の場合は次に使用可能なバイト境界にマップするよう指定するためのものである。

一般形式：

ALIGNED | UNALIGNED

一般規則：

1. ALIGNED属性とUNALIGNED 属性は，本来要素データの属性であるが，任意の配列および構造にこれらの属性を適用することもできる。配列ないし構造にこのいずれかの属性を適用するということは，その中に含まれている要素のうち，ALIGNED 属性またはUNALIGNED 属性が明示的に宣言されていないすべての要素にこの属性を適用することと同じである。

2. ある構造が配列ないし他の構造を含んでいる場合に，含まれている配列ないし構造にALIGNED または UNALIGNEDのいずれかの属性が適用されると，この属性が活かされ，この配列ないし構造に適用されるはずであった，含んでいる構造のALIGNED属性または，UNALIGNED 属性は無効になる。

3. LIKE構造変数の場合は，その含まれている要素にALIGNED属性およびUNALIGNED属性が適用される前に，LIKE属性が展開される。LIKE構造変数からとられるALIGNED 属性およびUNALIGNED 属性は，その構造変数の小構造および要素に対して明示的に指定されているものだけである。

4. ビット・クラス・データおよび文字クラス・データが関係しているオーバーレー定義づけにおいては，定義された項目も基底項目のオーバーレーされる部分も共にUNALIGNEDでなければならない。これ以外のすべてのタイプの定義においては，同義の項目は共にALIGNEDであるか，共にUNALIGNED でなければならない。

5. 実際に渡される引数のALIGNED属性および UNALIGNED属性は，対応するパラメーターの属性と一致していなければならない。もとになる引数の属性が対応するパラメーターの属性と合致していない場合には，見かけ引数が作り出される。

6. 基底付変数が他の変数のある世代を参照するのに使用されている場合は，この双方の変数のALIGED属性と UNALIGNED属性は一致していなければならない。

7. ストリング・データの境界合せは，与えられている ALIGNED属性あるいはUNALIGNED属性によって左右されるだけでなく，このストリングが固定長であるか可変長であるかによっても左右される。図K．1 および，図K.2 に，ストリングの境界合せについてのまとめが載っている。

8. EVENT 属性あるいはAREA 属性をもつデータに UNALIGNED 属性を与えることはできない。

9. UNALIGNED 属性をもつ固定長ビット・ストリングがADDR 関数の引数として使用されている場合，または，LOCATEないしALLOCATEステートメントの中で使用されている基底付構造の最初の要素としてUNALIGNED 属性をもつ固定長ビット・ストリングが使用されている場合は，返されたロケーター値がそのビット・ストリングの最初のビット位置を指していないことがある。

省略時属性：

1. 省略時属性は要素のレベルで適用される。ビット・ストリング・データ，文字ストリング・データ，および数字データの省略時属性はUNALIGNED である。これ以外のタイプのデータの省略時属性は，すべてALIGNED である。

2. すべての演算子，および組込み関数ならびにユーザー定義の関数に関して，その結果の要素には省略時属性として ALIGNED またはUNALIGNEDが適用される。

3. 定数には，ALIGED またはUNALIGNEDの省略時属性が適用される。

AREA

AREA 属性は，割振りをする際に，基底付変数の割振りのために確保しておくべき記憶域を定義するためのものである。このようにして確保されている記憶域に関しては，ALLOCATE ステートメントおよびFREEステートメントのINオプションでそのエリア変数を指名することにより，これを基底付変数に割り振ることも，またこれを基底付変数から解放することもできる。解放された記憶域を再び基底付変数に割り振ることもできる。

一般形式：

AREA 〔（大きさ）〕

一般規則：

1. 確保すべきバイト数を指定する整数値をもつ式によって，エリアの大きさを与える。

2. 静的記憶域クラスのエリアの大きさを指定する場合には，10進整数定数で指定しなければならない。理論上許される最大の大きさは，16,777,200である。実際には，上限は使用可能な記憶域の大きさによって決まる。

3. 宣言されているエリア変数が被制御変数かパラメーターである場合には，エリアの大きさとして星印を指定することができる。星印が指定されて宣言されている被制御エリア変数の場合は，このエリアを割り振るために用いられるALLOCATE ステートメントの中でエリアの大きさを指定しなければならない。星印が指定されて宣言されているパラメーターの場合には，引数からエリアの大きさが引き継がれる。

4. エリア・タイプのデータを他のタイプに変換することはできない。また，エリアは，エリア変数に対してしか割り当てることができない。

5. 演算子をエリア変数に適用することはできない。

6. エリア変数にUNALIGNED属性を与えることはできない。

7. エリアがBASED属性をもつ場合，大きさの属性は10進整数定数でなければならない。ただし，このエリアが基底付構造のメンバーであり，しかもREFERオプションが使用されている場合にはこの限りでない（「第8章： 記憶域の制御」を参照のこと）。

8. RECORD 入出力が行なわれている場合，エリアのエクステントと制御情報だけが転送される。

省略時属性：

1. 大きさが指定されていない場合，省略時の大きさがとられる。当コンパイラーに関しては，この省略時の大きさは1000バイトである。

2. OFFSET属性またはINオプションの中にエリア変数があると，このエリア変数は文脈宣言されたことになる。

AUTOMATIC，STATIC，CONTROLLED，BASED

省略形： AUTOMATICに対してはAUTO
CONTROLLEDに対してはCTL

記憶域クラスの属性は，データ変数の記憶域割振りのタイプを指定するために使用される。

AUTOMATICは，記憶域宣言がその内部にあるブロックに入るたびに記憶域を割り振るよう指定するものである。このブロックから外に出ると，記憶域は解放される。このブロックが回帰的に呼び出されるプロシージャーである場合には，このブロックに入ると，先に割り振られていた記憶域が1つの層ずつ「下に押下げられる」。また，回帰的プロシージャーの各世代が終了するたびに，最も新しく割り振られた記憶域が「上に押上げられる」（積重ねの各層の押上げと押下げについては，「第6章： プログラムの構成」を参照のこと）。

STATICは，当該プログラムをロードするとき記憶域を割り振り，この後プログラムの実行が完了するまでこの記憶域を解放しないよう指定するためのものである。

CONTROLLEDは，プログラマーがALLOCATE ステートメントおよびFREEステートメントを使って記憶域の割振りならびに解放を完全に制御することを指定するためのものである。同一の被制御変数を，間で解放せずに2回以上続けて割り振ると（複数割振り），その変数の各世代が積み重ねられることになる。

BASEDは，CONTROLLED と同様，プログラマーの側で記憶域の割振りならびに解放を完全に制御することを指定するためのものである。ただし，この制御方法は多様である。この点については，「第8章： 記憶域の制御」で説明されている。複数割振りを行なっても，各世代は積み重ねられず，いつでも各世代を使用することができる。すなわち，ロケーター変数の値によって特定の世代を識別することができるわけである。

一般形式：

STATIC|AUTOMATIC|
CONTROLLED|
BASED〔（要素ロケーター式）〕

一般規則：

1. 自動変数および基底付変数の有効範囲は内部のみである。静的変数および被制御変数の有効範囲は内部でも外部でもよい。

2. 入口定数，ファイル定数，構造のメンバー，およびDEFINEDデータ項目には，記憶域クラスの属性を与えることができない。

3. パラメーターに対しては，記憶域属性CONTROLLEDを明示的に宣言することはできるが，STATIC，BASEDまたはAUTOMATIC の各属性は宣言することはできない。

4. 調節可能な長さと次元をもつものとして宣言されている変数にSTATIC属性を与えることはできない。

5. 構造変数に関しては，大構造名に対してだけ記憶域クラスの属性を与えることができる。大構造名に記憶域クラスの

属性を与えると，その構造のすべての要素，または構造の配列全体にこの属性が適用される。
ある構造にCONTROLLED属性またはBASED属性を与えた場合は，プログラマーの側で割り振ったり解放したりすることができるのは大構造だけであり，要素の割振りや解放をすることはできない。

6. 基底付変数を使用するときは，以下の規則が適用される。

a. ある基底付変数の参照を完成するためにロケーター値が必要であるが，ロケーター値が1つも明示的に指定されていないというときはつねに，関連するBASED属性内のロケーター式の値が使用される。ロケーターが1つも宣言されていなければエラーになる。

b. 基底付変数が参照される場合，修飾しているポインター変数がデータの所在場所を示すのに対して，とられるデータ属性は，その基底付変数のデータ属性である。

c. 次のような用途に基底付変数を使用することができる。既存のデータを識別し，そのデータについて記述する。ALLOCATEステートメントによって記憶域を取得する。LOCATEステートメントを用いて出力緩衝域の中の記憶域を取得する。

d. あるエリアの中に割り振られている基底付変数のエリア内における相対的な位置は，オフセット変数の値によって識別される。

e. 基底付変数の宣言の中でEXTERNAL と指定することはできない。しかし，外部ポインター変数によって基底付変数参照を修飾することはできる。

f. REFERオプションを使用することによって，調節可能なエリアの大きさ，配列の限界，およびストリングの長さを含めて基底付構造を宣言することができる。「第8章： 記憶域の制御」を参照のこと。

g. CHECK接頭語リストの中の基底付変数への参照，またはデータ指示入出力用データ・リストの中の基底付変数への参照を，ロケーターで明示的に修飾することはできない。

h. BASED VARYING ストリングの最大の長さは，重ねて定義される側のストリングの最大の長さと等しくなければならない。たとえば，

```
DECLARE A CHAR(50) VARYING
                   BASED(Q),
        B CHAR(50) VARYING;
        Q=ADDR(B);
```

i. 基底付変数にINITIAL 属性を与えることができる。この値は，ALLOCATEまたはLOCATE ステートメントでこの基底付変数を明示的に割り振るときにだけ使用される。同一のメンバーに対してREFER オプションとINITIAL属性とが使用されている場合には，REFERの目標にその値を割り当てた後で，初期設定が行なわれる。

省略時属性：

1. 内部変数の省略時記憶域クラスはAUTOMATIC であり，外部変数の省略時記憶域クラスはSTATIC である。

2. BASED属性の中，あるいは，LOCATE，ALLOCATE，ないしREADステートメントのSETオプションの中にポインター変数がある，または，ロケーター修飾語としてポインター変数が使われているという事はそのポインター変数が文脈的に宣言されたことを意味する。

BACKWARDS

BACKWARDS属性は，磁気テープ上のデータ・セットと結び付けられているSEQUENTIAL INPUTファイルのレコードを，逆の順序（すなわち，最後のレコードから始めて最初のレコードに至る順序）でアクセスするよう指定するためのものである。

一般形式：

BACKWARDS

一般規則：

1. BACKWARDS属性は，RECORDファイルに対してだけ適用される。すなわち，この属性はSTREAM属性とは両立しない。BACKWARDS属性はRECORDおよびSEQUENTIAL を暗示している。

2. BACKWARDS属性は，テープ・ファイルに対してだけ適用される。

BASED

AUTOMATIC を参照のこと。

BINARY，DECIMAL

省略形： BINARYに対してはBIN
DECIMALに対してはDEC

BINARY属性とDECIMAL 属性は，算術変数によって表わされているデータ項目の基数が，2または10であることをそれぞれ指定するためのものである。

一般形式：

BINARY|DECIMAL

一般規則：

BINARY属性またはDECIMAL属性を，PICTURE 属性と一緒に指定することはできない。

省略時属性：

宣言されていない識別名（または，次元属性，UNALIGNED 属性，ALIGNED 属性，有効範囲属性，記憶域クラス属性のいずれか1つまたはそれ以上の属性だけをもつものとして宣言されている識別名）は，その頭文字に応じて割り当てられる属性をもつ算術変数であるとみなされる。IからNの間の文字で始まっている識別名の標準省略時属性は，REAL FIXED BINARY(15,0) である。これ以外の英字で始まる識別名の標準省略時属性は，REAL FLOAT DECIMAL(6) である。FIXEDまたはFLOAT および／またはREAL またはCOMPLEX が宣言されている場合には，DECIMAL であるとみなされる。標準省略時精度は，システム／360およびシステム／370の各コンパイラーごとに定義されている精度である。

BIT，CHARACTER

省略形：　CHARACTERに対してはCHAR
　　　　　VARYINGに対してはVAR

BIT属性とCHARACTER属性は，ストリング変数を指定するのに使用される。BIT属性は，ビット・ストリングを指定するためのものである。CHARACTER属性は，文字ストリングを指定するためのものである。

一般形式：

```
{BIT      }
{         }  〔(長 さ)〕〔VARYING〕
{CHARACTER}
```

一般規則：

1. 長さ属性は，固定長ストリングの長さまたは長さの変わるストリングの最大の長さを指定するためのものである。「長さ」を指定していない場合には，1であるとみなされる。

2. VARYING 属性は，当該変数が長さの変わるストリングを表わしていることを指定するためのものであり，この場合には「長さ」は最大の長さを表わす。現在の長さはつねに現在の値の長さである。長さの変わるストリングに割り振られる記憶域は，宣言されている最大の長さより2バイトだけ長くなっている。この記憶域の最初の2バイトには，ストリングの現在の長さをバイト単位で表わす数字が入る。

3. 長さ属性を指定する場合には，共通属性の因数化について同じレベルで，CHARACTER属性またはBIT属性の直ぐ後に「長さ」を記入しなければならない（CHARACTERまたはBITと「長さ」の間には，ブランクを入れても入れなくてもよい）。

4. 式または星印を使って長さ属性を指定することができる。

   「長さ」が式である場合には，当該変数のための記憶域が割り振られるときにこの式が整数に変換される。データ集合体のためのパラメーター記述子の中で式を使用してはならない。

   パラメーターまたは被制御変数に対しては，「長さ」として星印を記入することができる。この場合は，先の記憶域割振りから長さをとることができるが，被制御変数に関しては後続のALLOCATE ステートメントで長さを指定することもできる。

5. ストリングがSTATIC 属性をもつ場合，その長さ属性は10進整数定数でなければならない。

6. ストリングがBASED属性をもつ場合には，このストリングが基底付構造のメンバーでありしかもREFER オプションが使用されている場合を除き，その長さ属性は10進整数定数でなければならない（「第8章： 記憶域の制御」を参照のこと）。

7. BIT属性，CHARACTER属性，およびVARYING 属性をPICTURE属性と一緒に指定することはできない。

8. CHARACTER のかわりにPICTURE属性を使って固定長文字ストリング変数を宣言することができる（PICTURE属性の項を参照のこと）。

9. ビット・ストリング変数および文字ストリング変数に許される最大の長さは，32,767 である。ストリングの最小の長さはゼロである。

BUFFERED，UNBUFFERED

省略形：　BUFFEREDに対してはBUF
　　　　　UNBUFFEREDに対してはUNBUF

BUFFERED属性は，補助記憶装置との間で転送が行なわれるときに，SEQUENTIAL RECORDファイルの各々のレコードが途中で緩衝域を経由しなければならないことを指定するものである。

UNBUFFERED属性は，このようなレコードが緩衝域を経由する必要がないことを指定するものである。ただし，この属性はこのようなレコードが緩衝域を経由してはならないと指定するものではない。実際には，UNBUFFEREDと指定されているときには，一般に，隠れている緩衝域が使用される。

一般形式：

```
BUFFERED|UNBUFFERED
```

一般規則：

BUFFERED属性またはUNBUFFERED 属性は，RECORD ファイルに対してだけ指定することができる。

省略時属性：

DIRECT ファイルの場合を除き，標準省略時属性はBUFFEREDである。DIRECTファイルの場合，標準省略時属性はUNBUFFEREDである。

注： 位置指定方式の入出力ステートメントであるLOCATE およびREAD SETを指定することができるのは，BUFFEREDファイルの場合だけであり，EVENTオプションを指定することができるのは，UNBUFFEREDファイルの場合だけである。

BUILTIN

BUILTIN 属性は，ある宣言で指定されている名前をその宣言の有効範囲内で参照したときは，すべて，同名の組込み関数，擬似変数，または組込みサブルーチンを指しているものと解釈するよう指定するためのものである。

一般形式：

BUILTIN

一般規則：

1. あるブロックが別のブロックに含まれている場合に，内側のブロックで組込み関数，擬似変数，または組込みサブルーチンの名前として用いられている識別名と同じものが，外側のブロックでは別の意味をもつと宣言されているときは，内側のブロックでBUILTINを使用する。

2. ある名前に対してBUILTIN属性が宣言されている場合には，INTERNAL属性が暗示される。この名前にそのほかの属性を与えることはできない。

3. パラメーターに対してBUILTIN属性を宣言することはできない。

注： 引数をもたない組込み関数（たとえば，ONCHAR，ONSOURCE，DATEなど）を使用する場合は，BUILTIN 属性をもつものとしてこれらの組込み関数を明示的に宣言するか，ナル・ストリング・リストを使ってこれらの関数を文脈的に宣言するか，またはDEFAULTステートメントを使ってこれらの関数を暗示的に宣言するかしなければならない。

CHARACTER

BITの項を参照のこと。

COMPLEX，REAL

省略形： COMPLEXに対してCPLX

COMPLEX 属性とREAL 属性は，算術変数のモードを指定するのに使用される。REAL 属性は，当該変数によって表わされるデータ項目が実数であることを指定するものである。COMPLEX属性は，当該変数によって表わされるデータ項目が複素数であり，したがって，各データ項目が2つの要素から構成されている（最初の要素が実数で2番目の要素が虚数），ことを指定するものである。

一般形式：

REAL | COMPLEX

一般規則：

数字変数が複素数値を表わす場合には，PICTURE 属性と一緒にCOMPLEX属性を指定しなければならない。PICTURE 属性と一緒に指定することのできる算術データないしストリング・データの属性は，COMPLEX属性だけである。

省略時属性：

標準省略時属性はREALである。

CONDITION

省略形： COND

CONDITION属性は，当該宣言で指定されている識別名が条件名であることを指定するためのものである。

一般形式：

CONDITION

一般規則：

1. 条件名に適用することのできる属性は，CONDITION 以外には，有効範囲属性，すなわちINTERNAL またはEXTERNAL属性だけである。

2. 条件名を記入することができるのは，ON，SIGNAL，またはREVERTステートメントの中だけである。

省略時属性：

ON，SIGNAL，またはREVERTステートメントの中にCONDITION 条件をもつ識別名があると，この識別名は条件名であると文脈的に宣言される。

省略時の有効範囲はEXTERNALである。

CONNECTED

省略形： CONN

CONNECTED 属性は，パラメーターにだけ適用されるもので，

当該パラメーターが，連続している記憶域だけを参照するものであること，したがって，レコード方式の入出力において当該パラメーターを目標フィールドないし原フィールドとして使用したり，ストリングをオーバーレー定義する際に当該パラメーターを基底として使用したりすることができることを指定するのに用いられる。

一般形式：

CONNECTED

一般規則：

1. CONNECTEDは，CONTROLLED 属性をもたない集合体パラメーターの付加的属性であり，レベル－1の名前だけに与えることのできる属性である。この属性は，DECLARE ないしDEFAULT ステートメントの中，または，ENTRY属性のパラメーター記述子の中で指定することができる。

2. CONNECTED属性をもつパラメーターに渡される引数が連続していない場合には，連続している記憶域の中に見かけ引数が作り出される。

CONTROLLED

AUTOMATICの項を参照のこと。

DECIMAL

BINARYの項を参照のこと。

DEFINED

省略形：　DEF

DEFINED 属性は，ここで宣言されている変数を，指定されている基底変数と結びついている記憶域の一部ないし全体と結びつけるよう指定するためのものである。

一般形式：

DEFINED{基底変数|(基底変数)}
〔POSITION(要素式)〕

「基底変数」は，ここで宣言されている変数と結びつけられる記憶域をもつ変数である。ここで宣言されている変数は，「被定義変数」である。

POSITION 属性は，ストリング基底変数内の被定義変数と結びつけられる部分の最初の位置を指定するものである。この位置は，基底変数において必要とされる部分の最初のビットないし文字の位置である。

一般規則：

1. ある変数を他の変数の上に重ねて定義する目的は，内部に貯えられているデータをプログラマーが2つ以上の名前で参照することができるようにすることである。基底変数の名前は，そのデータに対して初めに宣言された名前である。この基底変数に重ねて定義される各々の変数は，それぞれ異なる名前をもつ。内部に貯えられているデータがデータの集合体である場合，被定義変数はそのすべてのデータを含んでいてもよいし，その特定の部分だけを含んでいてもよい。被定義変数は，基底変数の属性を引き継がない。

2. 定義には，単純定義，iSUB 定義，ストリング・オーバーレー定義の3つのタイプがある。

   POSITION属性が指定されている場合，この定義はストリング・オーバーレー定義である。この場合には，基底変数にiSUB 参照が含まれていてはならない。基底変数で指定されている添字にiSUB 変数への参照が含まれている場合，この定義はiSUB 定義である。iSUB 変数もPOSITION属性もない場合，基底変数と被定義変数とが下に示す基準の通りに一致しているならば，これは単純定義であり，一致していないならばストリング・オーバーレー定義である。
   図I.3 はこの関係を図にまとめたものである。

POSITION 属性	指定されている	指定されていない		
基底変数の添字の中でのiSUB変数への参照	—	ある	ない	
基底変数と被定義変数との関係[1]	—	—	一致する	一致しない
定義のタイプ	ストリング・オーバーレー定義	iSUB 定義	単純定義	ストリング・オーバーレー定義

[1] 一般規則の第2項を参照のこと。

図I.3　定義のタイプ

基底変数が，引数として渡されるときに，被定義変数の(DEFINED 属性以外の)諸属性をもつパラメーターと一致している場合は，基底変数と被定義変数とは一致しているという。この際，このパラメーターの配列の限界，ストリングの長さ，およびエリアの大きさはすべて星印によって指定されているものとみなされる。

単純定義およびiSUB定義の場合，あるPICTURE属性と一致させることができるのは，繰返し因数以外は全く同じであるようなPICTURE属性だけである。ストリング・オーバーレー定義の場合は，参照によって有効な基底変数を指定するには，この参照は連続している記憶域に対するものでなければならない。

実施にあたって，プログラマーの側で上で述べた一致に関する規則をことごとく変更することができる。ただし，これを変更すれば，望ましくない波及効果がプログラマー自身のプログラムに及ぶ可能性があることを覚悟しておかなければならない。

3. 被定義変数の中で，配列の限界の値，ストリングの長さの値，またはエリアの大きさの値について指定されているか，推定される値は，必ずしも基底変数の対応する値と一致している必要はない。しかし，定義される配列，ストリングまたはエリアは，これと対応する基底の配列，ストリングまたはエリアに納まるものでなければならない。

4. 基底変数の属性の中には，被定義変数の属性と一致する必要のないものまたは一致し得ないものがある。基底変数および被定義変数に次のような制約があるからである。

   基底変数：

   許される属性　EXTERNALまたはINTERNAL
   修飾されているか，または添字がついている，またはその両方
   パラメーター（ストリング・オーバーレー定義においては，パラメーターは連続している記憶域を参照するものでなければならない）

   許されない属性　BASED
   DEFINED

   被定義変数：

   必ず備えていなければならない属性
   INTERNAL
   レベル－1の識別名

   他の許される属性
   次元属性

   許されない属性　INITIAL
   AUTOMATIC/BASED/CONTROLLED/STATIC
   パラメーター

5. 基底変数がEXTERNAL属性をもつ場合は，被定義変数がその内部にあるプロシージャーの中でこの基底変数が知られていなければならない。EXTERNAL 基底変数がいくつかの外部プロシージャーの中で知られていることがある。この場合，この中のある1つのプロシージャーにおいてこの変数の値が変更されると，被定義変数の値も同様に変更される。

6. 被定義データを参照した場合は，基底変数ではなしに被定義変数の配列の限界およびストリングの長さに関して，SUBSCRIPTRANGE条件およびSTRINGSIZE条件が引き起される。

7. 値の確定および名前の解釈は，次の順序で行なわれる。

   a. 被定義変数が宣言されているプロシージャーに入ると，この変数の配列の限界，ストリングの長さ，およびエリアの大きさが計算される。

   b. 被定義変数への参照は，基底変数の現在の世代への参照である。見かけ引数が作成されずに被定義変数が引数として渡される場合，対応するパラメーターが参照するのは，この引数が渡される時点の基底変数の世代である。呼び出されたプロシージャーの中で基底変数が再割振りされている場合にもこの点に変りはない。

   c. 被定義変数への参照が行なわれるときは，被定義変数の中の添字の値が計算され，この値が整数に変換され，その後で基底変数の中の添字の値が計算される。

単純定義

単純定義を行なうと，要素変数，配列変数，または構造変数を別の名前で参照することができるようになる。

一般規則：

1. 被定義変数および基底変数は，任意のデータ・タイプのデータを含むことができるが，先に述べた意味でこの2つは一致していなければならない。被定義変数の中の要素に対してALIGNED属性ないしUNALIGNED属性が指定されている場合には，基底変数の中の対応する要素に対しても同様にALIGNED属性ないしUNALIGNED属性が指定されていなければならない。

2. 被定義変数には次元属性を与えることができる。基底変数には添字をつけることができる。ただし，この添字がiSUB変数であってはならない。

3. 単純定義においては，POSITION属性を指定することはできない。

4. 配列の単純定義においては，

a． 基底変数は，配列の断面であってもよい。

b． 被定義変数の次元数は，基底変数の次元数に等しくなければならない。

5. ストリングの単純定義においては，被定義ストリングの長さは，基底ストリングの長さと同じかそれ以下でなければならない。

6. エリアの単純定義においては，被定義エリアの大きさは，基底エリアの大きさと同じかそれ以下でなければならない。

7. 基底変数は，VARYINGストリングであってもよいし，VARYING ストリングを含んでいてもよい。ただし，その場合は，被定義変数の対応する部分がこのストリングの最大の長さと同じかそれ以下の VARYING ストリングでなければならない。

例：

```
DCL A(10,10,10),
    X1(2,2,2) DEF A,
    X2(10,10) DEF A(*,*,5),
    X3 DEF A(L,M,N);
```

X1は，Aの各行，列，および面の最初の2つの要素からなる3次元配列である。X2は，Aの5番目の面からなる2次元配列である。X3は，添字式L，M，Nによって識別される要素からなる要素である。

```
DCL B CHAR(10),
     Y CHAR(5) DEF B;
```

Yは，Bの最初の5文字からなる文字ストリングである。

```
DCL C AREA(500),
     Z AREA(500) DEF C;
```

Zは，Cに重ねて定義されるエリアである。

```
DCL 1 D UNALIGNED,
          2 E,
          2 F,
               3 G CHAR(10) VAR,
               3 H,
     1 S UNALIGNED DFF D,
          2 T,
          2 U,
               3 V CHAR(10) VAR,
               3 W;
```

Sは，Dに重ねて定義される構造である。単純定義においては，この2つの構造の組立て方が同じでなければならない。Tへの参照はEへの参照であり，Vへの参照はGへの参照である。他の参照も同様である。

iSUB定義

iSUB 定義を行なうと，プログラマーは基底配列内の指定した要素からなる被定義配列を作成することができる。被定義配列あるいは基底配列が構造の配列であってもよい。

一般規則：

1. 被定義配列も基底配列も任意のデータ・タイプのデータを含むことができる。双方の配列は，（次元属性を除き）同じ属性をもつものでなければならない。

2. 被定義変数には次元属性を与えなければならない。基底変数には添字をつけてもよい。ただし，添字を指定するのに星印を使用することはできない。

3. iSUB 定義においては，POSITION属性を指定することはできない。

4. iSUB 変数は，基底配列のための添字リストにおける，被定義配列のi次元への参照である。基底の配列添字リストの中の少なくとも1つの添字は，値の計算の後，基底配列内の所要の添字を与えるiSUB 式でなければならない。iの値の範囲は，1からnまでである（nは被定義配列の次元数）。基底変数の添字の数は，基底配列の次元数に等しくなければならない。

5. 値の計算に関する一般規則の外に，以下の点に注意しなければならない。

a． 被定義配列への参照において添字式が指定されていない場合は，この参照を含む式ないし割当ての値が計算されている間に添字の値が計算される。

b． iの値は，10進整数定数として指定される。iSUB式においては，iSUB変数は省略時精度の固定2進変数として取り扱われる。

c． 基底の変数添字リストの中に対応するiSUB がない場合にも，被定義変数への参照の中の添字の値が計算される。

6. iSUB 被定義配列が引数として渡される場合，確実に見かけ引数が作成されるようプログラマーの側で特に配慮しなければならない。たとえば，この引数を括弧で囲んだり，対応するパラメーターにCONNECTED属性を与えたりしなければならない。iSUB 被定義配列の要素の場合は，見かけ引数を作成することなく引数として渡すことができる。

例：

```
DCL A(100,100) CHAR(1),
    X(10,10) CHAR(1)
     DEF A(1SUB+20,2SUB+90);
```

Xは，第1次元がAの限界21－30の間の要素，第2次元がAの限界91－100の間の要素からなる2次元配列である。

```
DCL B(2,5),
    Y(5,2) DEF B(2SUB,1SUB);
```

Yは，限界を入れ替えたBの要素からなる2次元配列である。

```
DCL A(10,10)
    B(5,5) DEF A(1+1SUB/5,
                 1+2SUB/5);
```

この場合には，Aのある1つの要素にBのいくつかの要素が重ねて定義される。
B(I，J)は次の要素に重ねて定義される。

I＜5かつJ＜5ならば，A(1，1)
I＜5かつJ＝5ならば，A(1，2)
I＝5かつJ＜5ならば，A(2，1)
I＝5かつJ＝5ならば，A(2，2)

I＜5かつJ＜5ならば要素B(I，J)はすべてA(1，1)に重ねて定義されるから，これらの要素の内の1つに値を割り当てると，このすべての要素に同じ値が割り当てられることになる。

ストリング・オーバーレー定義

ストリング・オーバーレー定義を行なうと，プログラマーは，ある被定義変数を，基底変数のための記憶域に結びつけることができる。被定義変数も基底変数も共にストリング・データないしピクチャー・データでなければならない。

一般規則：

1. 被定義変数も基底変数も，ALIGNED 属性または VARYING属性をもつことはできない。

2. 被定義変数と基底変数は共にビット・クラスに属しているか，共に文字クラスに属していなければならない。ビット・クラスに属するものは，以下の通りである。

   a. 固定長ビット・ストリング

   b. 固定長ビット・ストリングの集合体

   文字クラスに属するものは，以下の通りである。

   a. 固定長文字ストリング

   b. 文字ストリングと数値ピクチャー・データ

   c. aとbの集合体

3. ビット・クラス・データの集合体を配列の中の添字をもつ要素に重ねて定義することはできない。また，無符号の整数定数以外のものを含むPOSITIONオプションを用いて，ビット・クラス・データの集合体を定義することはできない。

4. iSUB 変数をストリング・オーバーレー定義における基底変数として使用することはできない。

5. 基底変数内の被定義変数が始まるビットないし文字を指定するのに，POSITION属性を使用することができる。このPOSITION属性は次の形式をとる。

   POSITION(要素式)

   この式は，値の計算の後，基底変数の先頭を基準にした所要のビットないし文字の位置を与えるものである。この属性はDEFINED属性の前にあっても後にあってもよい。この属性を指定しなかった場合は，POSITION(1)とみなされる。この式によって与えられる値は1からnの間の数である。ここで，nは次のように定義される。

   n＝N(b)－N(d)＋1

   ここで，N(b)は，基底変数のビット数ないし文字数であり，

   N(d)は，被定義変数のビット数ないし文字数である。

   被定義項目への参照があるたびに，この式の値が計算され，その値が整数に変換される。この値の許容最大値は32,767である。

6. 基底変数は連続している記憶域の中のデータを参照するものでなければならない。

```
DCL  A CHAR(100),
     V(10,10) CHAR(1) DEF A;
```

Vは文字ストリングAのすべての要素からなる2次元配列である。

```
DCL  B(10) CHAR(1),
     W CHAR(10) DEF B;
```

Wは配列Bのすべての要素からなる文字ストリングである。

```
DCL  C(10,10) BIT(1),
     X BIT(40) DEF C POS(20);
```

Xは，Cの20番目以降の40の要素からなるビット・ストリングである(20番目の要素を含む)。

```
DCL E PIC'99V.999',
    Z1(6) CHAR(1) DEF E,
    Z2 CHAR(3) DEF E POS(4),
    Z3(4) CHAR(1) DEF E POS(2);
```

Z1は，10進数ピクチャーEのすべての要素からなる文字ストリング配列である。Z2は，ピクチャーEの要素'999'からなる文字ストリングである。Z3は，ピクチャーEの要素'9.99'からなる文字ストリング配列である。

次元属性

次元属性は，配列の次元数ならびに各次元の限界を指定するためのものである。次元属性では，限界(上限または上限と下限)を指定するか，または当該配列の実際の限界がどこか他の場所からとられることを星印を使って表示する。

一般形式：

(限　界〔,限　界〕・・・)

ここで，「限界」は，

{〔下　限：〕上　限}|*

であり，「上限」と「下限」は要素式である。

一般規則：

1. 指定されている限界の数は，当該配列の次元数を表わす。ただし，ここで宣言されている変数が構造の配列の中に含まれている場合は別で，この場合には，含んでいる構造から次元が引き継がれる。

2. 指定されている限界にもとづいて，実際の限界は次のように定められる。

   a. 上限だけが指定されている場合には，下限は1であるとみなされる。

   b. 下限は，上限と同じかそれ以下でなければならない。

   c. 星印は，次のような意味を表わす。ここで宣言されている変数がCONTROLLED属性を与えられている場合は，実際の限界はALLOCATEステートメントの中で指定される。この変数が単純パラメーターである場合は，実際の限界はこのパラメーターと対応する引数の宣言の中で指定される。このことからも分るように，星印を記入することができるのは，パラメーターおよびCONTROLLED変数の場合だけである。

3. 限界として指定されている式は，配列に記憶域が割り振られるときに計算され，FIXED BINARY(15,0)に変換される。単純パラメーターの場合，限界は10進整数定数(符号はあってもなくてもよい)または星印でなければならない。

4. STATICと宣言されている配列の限界は，10進整数定数(符号はあってもなくてもよい)でなければならない。

5. BASEDと宣言されている配列の限界は，10進整数定数(符号はあってもなくてもよい)でなければならない。ただし，この配列が基底付構造の一部であり，しかもREFERオプションが使用されている場合にはこの限りでない。(「第8章： 記憶域の制御」を参照のこと。)

6. 次元属性は配列名(または，共通属性が因数化されている場合は，括弧でくくられた名前のリスト)の直ぐ後に置かなければならない。両者の間にはブランクがあってもなくてもよい。

7. 許容される最大の次元数は15である。下限の許容最小値は−32,768である。上限の許容最大値は32,767である。

DIRECT，SEQUENTIAL

省略形：　SEQUENTIALに対してSEQL

DIRECT属性とSEQUENTIAL属性は，RECORDファイルのレコードへのアクセス方法を指定するためのものである。SEQUENTIAL属性は，当該データ・セットにおけるレコード

の物理的な順序または論理的な順序に従ってレコードにアクセスするよう指定するためのものである。DIRECT属性は，キーを使って，当該ファイルのレコードをアクセスするよう指定するためのものである。したがって，直接ファイルの中の各々のレコードはあるキーと結び付いていなければならない。なお，いずれの属性もRECORD属性を暗示している。

SEQUENTIAL属性もDIRECT属性も，ファイルの現在の使用方法を指定するだけであって，当該ファイルと結び付いているデータ・セットの物理的な性質を指定するものではないということに注意していただきたい。実際には，SEQUENTIALファイルにおいて，データといっしょにキーが記録されていることもあるのである。また，DIRECTファイルは殆んどの場合，SEQUENTIALファイルとして作成される。

一般形式：

SEQUENTIAL|DIRECT

一般規則：

1. DIRECTファイルは，KEYED 属性をもっていなければならない。KEYED属性はDIRECT属性によって暗示される。SEQUENTIALファイルは，KEYED属性をもっていてもいなくてもよい。

2. DIRECT属性やSEQUENTIAL属性を，STREAM属性と一緒に指定することはできない。

省略時属性：

RECORD ファイルの省略時属性は，SEQUENTIALである。

ENTRY

ENTRY 属性は，ここで宣言されている識別名が外部入口定数または入口変数であることを指定するためのものである。また，ENTRY属性は，入口点のパラメーターの属性について記述するのにも使用される。

一般形式：

ENTRY〔（パラメーター記述子のリスト）〕

ここで，「パラメーター記述子のリスト」は次の形式をとる。

〔パラメーター記述子〔，パラメーター記述子〕・・・〕

パラメーター記述子のリストに関する規則

1. パラメーター記述子のリストは，結びつけられている外部入口定数または入口変数のパラメーターの属性について記述するためのものである。
パラメーター記述子のリストを与えない場合には，引数がパラメーターと合致しているものとみなされる。パラメーター記述子のリストが与えられている場合には，引数とパラメーターを突き合わせたり，見かけ引数を作成したりするのにこのリストが使用される。引数がパラメーターと合致していない場合には，パラメーター記述子のリストを与えなければならない。

2. 記述子は，1個のパラメーターの属性について記述するためのものである。たとえば，次のようなプロシージャーがあるものとしよう。

```
TEST:PROCEDURE (A,B,C,D,E,F);
        DECLARE A FIXED DECIMAL(5),
                B FLOAT BINARY (15),
                C POINTER,
                1 D,
                  2 P,
                  2 Q,
                    3 R FIXED DECIMAL,
                1 E,
                  2 X,
                  2 Y,
                    3 Z,
                F(4) CHARACTER (10);
                    .
                    .
                    .
        END TEST;
```

このプロシージャー内のパラメーターの記述子は，次のように宣言することができるであろう。

```
DECLARE TEST ENTRY
          (DECIMAL FIXED (5),
          BINARY FLOAT  (15),
          ,
          1,
            2,
            2,
              3 DECIMAL FIXED,
          (4) CHARACTER (10));
```

3. パラメーター記述子は，これらが記述しているパラメーターと同じ順序に並んでいなければならない。ある記述子が抜けている場合は，引数がパラメーターと合致しているものとみなされる。

4. あるパラメーターに対応する記述子が必要でない場合は，次のいずれかの方法でその記述子が省略されていることを明示しなければならない。

コンマを1つ記入する：
ENTRY(CHARACTER(10),,,FIXED DECIMAL) はパラメーターが4つあることを示している。

星印を1つ記入し，その後にコンマまたはパラメーター・リストの右括弧を記入する： ENTRY(*)はパラメーターが1つあることを示している。

カンマの後に記述子を記入せずにパラメーターがつづ

くときは，右括弧をとじて，記述子の省略を示す：ENTRY(FLOAT BINARY,)はパラメーターが2つあることを表わしている。

ENTRY( )と言う宣言はパラメーター記述子をもたないENTRY と等しいので，この入口名に引数を与えてはならない。

上の第3項に示した例では，パラメーターCも構造パラメーターEも記述子をもたない。

5. 一般に，パラメーター記述子の中では，任意の順序に属性を記入することができる。しかし，配列パラメーターの記述子の場合は，最初に次元属性を記入しなければならない。また，構造パラメーターの記述子の場合は，対応するパラメーターのレベル番号と同じ順序にレベル番号を記入しなければならず，各レベルの属性を記入する前に各々のレベル番号を記入しなければならない。記述子のレベル番号はパラメーターのレベル番号と同じである必要はないが，構造の組立て方は同じでなければならない。ある特定のレベルにおいては任意の順序に属性を記入することができる。

注： 記述子のレベル番号とそのレベルで指定されている属性はそれぞれ，カンマで区切られる(上の例を参照すること)。

6. パラメーター記述子の中に属性またはレベル番号が指定されていない限り，その記述子に省略時属性が適用されることはない。記述子の中にレベル番号および／または次元属性だけが指定されている場合は，FLOAT DECIMAL (6) REAL であるとみなされる。

7. パラメーター記述子の中でエクステント(長さ，大きさ，および限界)を指定するときは，10進整数定数または星印しか使用することができない。被制御パラメーターに対する記述子の中でエクステントを指定するときは，星印しか使用することができない。

8. DEFAULTステートメントとDESCRIPTORSオプションを結びつけて使用すれば，パラメーター記述子のリストで与える属性を暗示的に設定することができる。しかし，省略されている記述子には，これらの属性は適用されない。

一般規則：

1. RETURNS属性およびOPTIONS 属性は，パラメーター記述子のリストをもたないENTRY属性を暗示する。

2. BUILTIN属性またはGENERIC 属性と一緒にENTRY属性を指定することはできない。

3. 入口定数引数または入口変数引数を表わすパラメーターには，明示的または暗示的に，ENTRY属性を与えなければならない。ENTRY属性のネスティングについて許容される最大の深さは2である。たとえば，

```
DCL E ENTRY(ENTRY(FIXED))
```

は許されるが，

```
DCL E ENTRY(ENTRY(ENTRY(FIXED)))
```

は許されない。

4. ENTRY属性を指定するときに，パラメーター記述子のリストの中で共通属性の因数化を行なうことはできない。

5. 外部入口定数は，ENTRY属性をもつものとして明示的に宣言しなければならない。この明示的宣言を行なわないと，外部入口定数は，BUILTIN属性をもつものとして文脈的に宣言され，組込み関数の名前として取り扱われる。

6. 任意選択属性のVARIABLEは，付加的属性である。この属性が与えられていると，ここで宣言されている識別名が入口変数であると指定されることになる。次に挙げる属性のいずれかが宣言されていると，その識別名には，VARIABLE属性が暗示的に宣言される。

ALIGNED	次元
AUTOMATIC	INITIAL
BASED	パラメーター
CONTROLLED	STATIC
DEFINED	UNALIGNED

7. CALL ステートメントまたは関数参照の中で入口変数が使用されている場合は，対応する入口点は実行時までわからない。パラメーター記述子のリストをつけずに宣言された入口変数が，引数の受渡しを行なうCALLステートメントの中か，または，引数の受渡しを行なう関数参照として，出てきた場合は，引数と参照される入口点のパラメーターとが合致しているものとみなされる。しかし，入口変数の宣言の中でパラメーター記述子のリストが与えられている場合には，参照される入口点のパラメーターはパラメーター記述子のリストで与えられている属性をもつものとみなされ，必要に応じて，見かけ引数が作成される。

8. 入口式への参照に引数リスト(ナル引数リストであることもある)が含まれている場合は，つねに，この式によって表わされるプロシージャーが呼び出される。

9. 入口式への参照に引数リストが含まれていない場合は，次

のような文脈においては，この式によって表わされるプロシージャーは呼び出されない。

a．入口変数への割当ての右辺

b．入口式との比較。

c．総称入口名に渡される引数。

d．入口パラメーターに渡される引数。

e．UNSPEC組込み関数に渡される引数。

f．変数を必要とするすべての文脈（入口変数のみに該当する）。

10. CALLステートメントの中で使用される入口変数の値は，そのCALLステートメントが実行される時点において活用状態にあるブロックの入口点を表わしていなければならない。

11. 比較演算子＝または¬＝を使って2つの入口式の値を比較することができる。活用状態にないブロックの入口点を表わす値をもつ入口変数が比較演算の中で指定されていても，エラーではない。

12. 同一のPROCEDURE ステートメントまたは同一のENTRYステートメントの中にあるいくつかの入口名を比較しても，「等しい」という結果にはならない。

省略時属性：

暗示的にENTRY属性を指定することができる。PROCEDURE ステートメントまたはENTRYステートメントのラベル接頭語として識別名を記入すると，この識別名は入口定数として明示的に宣言されたことになる。この識別名の属性は，この明示的宣言と，さらに，DECLARE ステートメントで宣言が行なわれている場合には，この宣言からもとられる。次のようにして属性が取得される。

有効範囲属性：　外部入口定数はEXTERNAL属性をもつものとみなされる（INTERNAL属性は有効でない）。入口変数の省略時有効範囲属性は，INTERNALである。

RETURNS属性：　DECLARE ステートメントの中にRETURNS属性があれば，この属性がとられる。

ENVIRONMENT

省略形：　ENV

ENVIRONMENT属性は，PL／I言語に含まれていない種々のファイル特性を指定するために各コンパイラーごとに定義される属性である。

一般形式：

ENVIRONMENT （オプションのリスト）

「オプションのリスト」の中ではブランクまたはコンマによって個々のオプションを区切る。オプションのリストは，PL／Iの各コンパイラーごとに個々に定義される。次に示すのは当コンパイラーのオプションのリストである。

MEDIUM
〔FUNCTION〕
〔ASSOCIATE（ファイル名）〕
〔RCE〕
〔OMR〕
〔COLBIN〕
〔STACKER（n）〕
〔レコード形式〕〔BUFFERS（n）〕
〔データ・セットの編成〕
〔磁気テープの取扱い〕
〔印刷装置・穿孔装置の制御〕
〔COBOL〕〔データ管理の最適化〕
〔キーの文類〕
〔KEYLENGTH（n）〕
〔KEYLOC（n）〕
〔SCALARVARYING〕
〔直接アクセス装置の使用方法〕
〔ASCII －データ交換コード〕
〔BUFOFF〔（n）〕 － 緩衝域オフセット〕

これらのオプションはどのような順序で記入してもよい。これらのオプションの指定方法と用途については，「第10章：　入力と出力」，「第11章：　ストリーム方式の転送」，「第12章：　レコード方式の転送」を参照されたい。

一般規則：

ENVIRONMENT属性は，DECLARE ステートメントの中で記入しなければならない。OPEN ステートメントのオプションとして，この属性を指定することはできない。CLOSEステートメントのボリューム後始末オプションLEAVE に対するオプションとして，この属性を指定することができる。

EVENT

EVENT 属性は，この属性をもつ識別名が事象名として使用されることを指定するものである。事象名は，非同期入出力操作の現在の状態を調べるのに使用される。また，事象名をプログラム・スイッチとして使用することもできる。

一般形式：

EVENT

一般規則：

1. DECLARE ステートメントの中でEVENT 属性をもつものとして識別名を明示的に宣言することができる。WAITステートメント，DISPLAYステートメント，または種々の入出力ステートメントの中に識別名を記入することによって，その識別名に対してEVENT属性を文脈的に宣言することもできる（「第10章： 入力と出力」を参照のこと）。

2. 以下の属性を事象名に与えることができる。

   次元属性

   有効範囲属性（省略時属性はINTERNALである）

   記憶域クラス属性（省略時属性はAUTOMATICである）

   DEFINED 属性（事象名を他の事象名に重ねて定義することしかできない）

   INITIAL属性またはINITIAL CALL属性

3. 事象変数は，次のような2種類の値をもっている。

   a. この変数の完了値を表わす1つのビット。これが'1'Bであれば完了であり，'0'Bであれば未完了である。

   b. この変数の状況値を表わす精度が（15，0）の固定小数点2進数値。この値がゼロであれば正常な状況であり，ゼロ以外の値であれば異常な状況である。

   COMPLETION組込み関数およびSTATUS組込み関数を使用すると，事象変数のこの2つの値が別々に返される。COMPLETION組込み関数は，事象変数の完了値を表わすビット・ストリング値を返す。STATUS組込み関数は，事象変数の状況値を表わす固定小数点2進数値を返す。

   ある事象変数を他の事象変数に割り当てることは許される。しかし，事象変数と他のデータ・タイプのデータとの間で変換を行なうことはできない。

4. 事象変数はデータ集合体の要素であってもよい。また，事象データへの変換ないし事象データからの変換を伴なわない限り，事象変数を含んでいるデータ集合体を割当てに使用してもよい。

5. 以下のいずれかの方法によって，事象変数の値がセットされる。

   a. 完了値のセット ― COMPLETION擬似変数を使用する。

   b. 状況値のセット ― STATUS擬似変数を使用する。

   c. 事象変数の割当て。

   d. EVENT オプションをもつステートメントによって。

   e. 入出力事象ないしDISPLAY ステートメントと結び付いている事象変数を対象とするWAITステートメントによって。

   f. EVENTオプションを用いている入出力操作が，進行中のファイルを閉じることによって。

6. 割り振られた後の事象変数の状況値は，定義できない。

7. ステートメントの中でEVENTオプションを指定することによって，事象変数を入出力操作に結び付けることができる。この変数は，その事象が完了するまで事象と結び付けられている。入出力事象に関して言えば，事象変数と結び付けられている事象を対象とするWAITを実行している間にこの事象が完了する。この間，この事象変数は活用状態にあるという。活用状態にある事象変数を他の事象と結び付けたり，活用状態にある事象変数の完了値を事象変数の割当てによって変更したりするのは誤りである。

8. 入出力ステートメントを用いて，活用状態にある事象変数（配列，構造，またはエリアの中にある事象変数も含む）に値を割り当てるのは誤りである。

9. EVENTオプションをもつ入出力ステートメントが実行されると，対応する事象変数が非活用状態にあるならばその状況値がゼロにセットされる。この操作の間に割込みが起こることはなく，転送が行なわれる前に（しかし暗示的な開放に関連するすべての処置が完了した後）この操作は完了する。当該プログラムが終了するまでは，あるいは入出力事象変数の指定されているWAIT ステートメントが実行されるまでは，この入出力事象変数の完了値が「完了」にセットされることはない。WAIT は，この事象に関連する転送が終了するまで当該プログラムの実行を延期する。この操作に関連する入出力条件が引き起されなかったときは，この事象変数の完了値が「完了」にセットされ，この事象変数は活用状態でなくなる。入出力条件が引き起された場合には，事象変数の状況値が1にセットされた後で該当する条件が引き起される。これらの条件が引き起された結果いくつかのオン・ユニットに入りその最後のオン・ユニットから正常な状態で戻ると，あるいは，ある1つのオン・ユニットから異常な状態で戻ると，事象変数の完了値が

「完了」にセットされ，この事象変数は活用状態でなくなる。

10. 事象変数にUNALIGNED属性を与えることはできない。

EXTERNAL， INTERNAL

省略形： EXTERNALに対してはEXT
INTERNALに対してはINT

EXTERNAL属性とINTERNAL属性は，名前の有効範囲を指定するのに使用される。INTERNAL属性は，この属性をもつ名前が，この名前の宣言されているブロックおよびこれに含まれているブロックの中でだけ知られていることを指定するものである。EXTERNAL属性は，外部名としてその名前が宣言されている他のブロックにおいてもその名前が知られていることを指定するものである。

一般形式：

```
EXTERNAL | INTERNAL
```

一般規則：

1. 2つ以上のブロックにおいて大構造名をEXTERNAL として宣言する場合，どの宣言でもその構造のメンバーの属性は同じでなければならない（INITIAL属性を2回以上指定する必要はないからINITIAL属性だけは例外）。なお，対応するメンバー名は同じである必要はない。

2. 構造のメンバーはつねにINTERNAL属性をもっているので，有効範囲属性を与えることはできない。しかし，外部構造のメンバーを参照するときに，その参照が含まれているブロックに知られているメンバー名を使って参照すると，事実上，これと対応するメンバー名が同じであるか否かにかかわりなく，この外部名が知られているすべてのブロックにおいてこのメンバーを参照したことになる。

省略時属性：

内部プロシージャーの入口名の有効範囲属性，および記憶域クラスが与えられている変数の有効範囲属性は，INTERNAL とみなされる。ファイル定数の有効範囲属性，および外部プロシージャーの入口定数の有効範囲属性は，EXTERNAL とみなされる。プログラマーの定義になる条件名の有効範囲属性は，EXTERNAL とみなされる。

FILE

FILE属性は，ここで宣言されている識別名がファイル定数またはファイル変数であることを指定するものである。

一般形式：

```
FILE
```

一般規則：

1. ファイル変数には，ファイル属性（RECORD, INPUT など）を適用することはできない。

2. ファイル式とは，ファイル定数，ファイル変数，またはあるファイル値を表わす関数参照である。ファイル式には次のような用途がある。

   a. FILEオプションへの引数として使用する。

   b. 関数またはサブルーチンへ渡される引数として使用する。

   c. ON, SIGNAL, およびREVERTステートメントの入出力条件名への引数として使用する。

   d. RETURNステートメントへの引数として使用する。

   e. COPYオプションへの引数として使用する。

3. あるファイル定数の値を表わすファイル変数を使用することによって，そのファイル定数に対するオン・ユニットを設けることができる。たとえば，

```
DCL  F  FILE,
        G  FILE  VARIABLE;
        G=F;
L1:     ON  ENDFILE(G);
L2:     ON  ENDFILE(F);
```

   L1というラベルの ステートメントとL2というラベルのステートメントは，同じ指示を行なっている。

4. CALLステートメントまたは関数参照の引数としてファイル定数が指定されていると，これに代えて見かけ引数が作成される。

5. CHECK接頭語リストの中でファイル変数を指定することができる。CHECKリストの中にあるファイル変数に対するシステムの標準処置は，このファイル変数の識別名を印刷することである。ON, SIGNAL，およびREVERTステートメントのFILE オプションとしてこのような変数が出てきている場合は，この変数に対してCHECK条件は引き起されない。

6. レコード方式転送ステートメントによってファイル変数の値を転送することができる。しかし，転送後の値は有効でないこともある。

7. 比較演算子＝または¬＝を用いて2つのファイル式の値を比較することができる。2つの式が表わすファイル値がすべての部分について等しい場合にのみ，この比較の結果は等しいとされる。

省略時属性：

ファイル定数に「ファイル属性」のいずれかを与えると，このファイル定数に暗示的にFILE属性を与えることができる。 ファイ

ル属性については，「第１０章：　入力と出力」を参照されたい。また，入力ステートメントあるいは出力ステートメントのFILEオプションの中，または，入出力条件のためのONステートメントの中に識別名を記入することによって，その識別名をファイル定数として文脈的に宣言することができる。

FILE属性をもつ識別名が構造の要素であったり，以下のいずれかの付加的属性をもっている場合は，この識別名はファイル変数であるとみなされる。

記憶域クラスの属性
次元属性
パラメーター
ALIGNED または　UNALIGNED
DEFINED
INITIAL
VARIABLE

FIXED，FLOAT

FIXED属性とFLOAT属性は，ここで宣言されている算術変数のスケールを指定するものである。FIXED属性は，この変数が固定小数点データ項目を表わすことを指定するためのものである。FLOAT 属性は，この変数が浮動小数点データ項目を表わすことを指定するためのものである。

一般形式：

```
FIXED|FLOAT
```

一般規則：

PICTURE属性と一緒に，FIXED属性またはFLOAT 属性を指定することはできない。

省略時属性：

宣言されていない識別名（または，次元属性，ALIGNED 属性ないしUNALIGNED属性，有効範囲属性，記憶域クラス属性のうちいずれか１つ以上の属性だけをもつものとして宣言されている識別名）は，その頭文字に応じて決められる属性をもつ算術変数であるとみなされる。すなわち，この識別名がＩからＮの間の文字で始まっている場合，標準省略時属性はREAL FIXED BINARY（１５，０）である。これ以外の英字から始まっている場合，標準省略時属性はREAL FLOAT DECIMAL(6)である。BINARYないしDECIMAL および／またはREALないしCOMPLEXが指定されている場合には，FLOATであるとみなされる。標準省略時精度は，システム／３６０の実施上定義されている精度である。

FLOAT

FIXEDの項を参照のこと。

GENERIC

GENERIC属性は，一群の入口式の総称である入口名を定義するのに使用される。(その各々は，ここで宣言されている名前によって参照される）。この総称が参照されると，参照の中で総称に対して指定されている引数をもとにして１つの入口式が選ばれる。

一般形式：

```
GENERIC （入口式　WHEN
         （総称記述子のリスト）
         〔,入口式　WHEN
         （総称記述子のリスト）〕・・・）；
```

ここで，総称記述子のリストは次の形式をとる。

```
(〔記述子〔，記述子〕・・・〕)
```

一般規則：

1. GENERIC 属性が与えられる名前に対して指定できる属性は，INTERNAL属性だけである。

2. GENERIC の後に記入する入口式は各々，当該総称グループに属するメンバーに１つずつ対応している。この入口式は，入口定数または入口変数でなければならない。この入口式は，基底付，被定義，または添字付のものであってはならない。

3. 異なる記述子のリストをつけるのであれば，同じ入口式を１つのGENERIC宣言の中で２回以上記入してもよい。

4. 総称名への参照に含まれている引数にもとづいて，あるいは，総称名への参照に引数が全く含まれていないと言う事実にもとづいて，ある特定の入口式が選び出される。ある総称が参照されると，引数の数と各引数の属性が，左から右の順に各々の総称記述子のリストと比較され，ある総称記述子のリストのすべての属性が引数の属性と一致しているとわかったところでこの比較が終わる。その結果，この参照は，一致した総称記述子のリストをもつメンバーへの参照であると解釈される。

5. 許される属性は，特定の入口式の選定に関係するものだけである。そのような属性は，以下の通りである。

ALIGNED	
AREA	（大きさを指定することはできない）
基　底	
BIT	（長さを指定することはできない）
CHARACTER	（長さを指定することはできない）

ENTRY	(記述子のリストを指定することはできない)
EVENT	
FILE	
LABEL	(ラベルのリストを指定することはできない)
モード次元数	(限界を指定することはできない)
OFFSET	(エリア変数を指定することはできない)
PICTURE	'ピクチャー指定'
POINTER	
精度	(桁数を指定しなければならない。FIXEDの場合，さらに，スケール因数も指定しなければならない)
スケール	
UNALIGNED	
VARYING	

総称記述子のリストの中では，星印またはコンマによって記述子の省略を示すことができる。

6. ある総称値への参照の中で引数として使用されている入口式は，ENTRY タイプの記述子とだけ合致する。このような記述子がない場合，このプログラムは誤りである。

7. GENERIC 属性をもつ引数は，総称記述子のリストの中の ENTRY属性と合致する。

8. 総称記述子のリストの中でロケーター属性（POINTER またはOFFSET）が指定されている場合は，対応するパラメーターも同じ属性をもっていなければならない。この入口点が呼び出されたときに，一方のタイプから他方のタイプへの変換を行なうことはできない。

9. 総称記述子の中ではレベル番号を指定してはならない。集合体を引数として総称入口名に渡すことはできるが，見かけ引数は作成されない。

10. 総称名（参照ではない）を非総称入口名への引数として指定することができる。

この非総称入口名が入口変数または外部入口定数の場合には，パラメーター記述子のリストをつけてこれを宣言しなければならない。総称引数の記述子は，パラメーター記述子のリストをもつENTRYでなければならない。このようにネストされたリストは，受け渡す引数を選定するのに使用される。たとえば，

```
A: PROC;
   DCL B GENERIC (C WHEN
                    (FIXED),
                  D WHEN
                    (FLOAT)),
       E ENTRY (ENTRY(FIXED));
   CALL E(B);
     ⋮
   END A;
```

プロシージャーEが呼び出されると，Cが選び出され，引数として渡される。これは，入口名パラメーターの指定するパラメーターがFIXEDであると，記述子によって指定されているためである。

非総称入口名が内部入口定数の場合には，パラメーター記述子のリストをもつENTRYとして，対応するパラメーターを宣言しなければならない。このリストは，受け渡す引数を選び出すのに使用される。たとえば，

```
A: PROC;
   DCL B GENERIC (C WHEN(FIXED),
                  D WHEN(FLOAT));
   CALL E(B);
   E:  PROC(P);
       DCL P ENTRY(FIXED);
         ⋮
       END E;
   END A;
```

プロシージャーEが呼び出されると，Cが選び出され，引数として渡される。これは，入口名パラメーターがFIXEDであると宣言されているためである。

<u>INITIAL</u>

省略形：　INIT

INITIAL 属性には2つの形がある。第1の形は，あるデータ項目に記憶域が割り振られるときにそのデータ項目に割り当てる値をもつ初期定数，式，または関数参照を指定するためのものである。第2の形は，割振りの際に初期設定を行なうために，CALLオプションを介して，あるプロシージャーを呼び出すことを指定するためのものである。呼び出されたルーチンを実行している間に割当てによって変数の初期設定が行なわれる（このルーチンは呼出し点に値を返す関数として呼び出されるものではない）。

一般形式：

1. INITIAL （項目〔，項目〕...）

2. INITIAL CALL入口式
   〔引数のリスト〕

一般規則：

定数，被定義データ，構造，またはパラメーター（CONTROLLEDパラメーターを除く）にINITIAL 属性を与えることはできない。

第1形のINITIAL属性に関する規則：

1. リストの中の各々の項目は，定数，括弧でくくられた式，参照，星印（ある特定の項目の初期設定を行なわないことを示す），または反復指定のいずれかの形をとることができる。

2. この説明においては，「定数」は次のいずれか1つを表わす。

〔＋｜－〕算術定数

文字ストリング定数

ビット・ストリング定数

〔＋｜－〕実定数｛＋｜－｝虚定数

ラベル定数

入口定数

ファイル定数

「式」は，初期設定されるデータ項目に割り当てられる初期値を与えるための要素式を表わす。INITIAL 属性の中で式を指定する場合には，つねに，式を括弧で囲む。「参照」は，データ項目の初期値として使用することのできる変数または関数への参照を表わす。

3. INITIAL 属性が適用される時点は，当該変数の記憶域クラスによって異なる。

STATIC: 当該変数が宣言されている外部プロシージャーに入ったとき。

AUTOMATIC: 当該変数が宣言されているブロックに入ったとき。

CONTROLLED: ALLOCATE ステートメントが実行されたとき。

BASED: 当該変数を対象とするALLOCATEステートメントが実行されたとき。当該変数がポインターをセットすることによって参照されるもので，ALLOCATE ステートメントの中では指定されていないという場合は，DECLARE ステートメントの中でINITIAL 属性が指定されていてもこの属性は適用されない。

4. 1つの要素変数に対して，初期値は1つしか指定することができない。配列変数に対しては，2つ以上の初期値を指定することができる。構造変数を初期設定するときは，その基本項目の名前（これが要素変数であっても配列変数であっても）を個々に初期設定しなければならない。

5. ある配列に対して指定されている初期値は，この配列の要素に，行を主とした順序で（最後の添字が最も速く変化する），連続して割り当てられる。

6. ある配列に対して指定されている初期値の数が多すぎる場合には，余分の初期値は無視される。指定されている初期値の数が不足している場合には，配列の残りの部分は初期値化されない。

7. STATIC変数に対しては，INITIAL 属性の中で定数値しか指定することができない。ただし，STATICポインター変数を初期値化するときは，NULL 組込み関数を使用することができる。

8. 反復指定は，次の一般形式のいずれか1つをとる。

（反復因数）
参 照 ｜ 定 数 ｜（式）

（反復因数）
項 目〔，項 目〕・・・

（反復因数） ＊

「反復因数」は，配列の要素を初期値化するときに繰返し使用すべき定数，式，または項目リストの繰返し回数を指定するためのものである。反復因数の後に定数または式が記入されている場合には 指定されている数だけの要素が，この値によって初期値化される。反復因数の後に項目のリストが記入されている場合には，各項目が配列の要素を1つずつ初期値化しながら，指定されている回数だけこのリストが繰り返される。反復因数の後に星印が記入されている場合には，初期値化の際に，指定されている数だけの要素がスキップされる。

9. STATICデータの場合を除き，反復因数は要素式でもよい。STATICデータの場合には，反復因数は無符号の10進整数定数でなければならない。配列に対して記憶域が割り振られるときに，式の値が計算され，繰返し回数を表わす整数が与えられる。

10. 反復因数として負の値またはゼロを指定すると，初期値化は行なわれない。

11. ストリングからなる配列の初期値化では，ストリング繰返し因数と反復因数の両方を含めてもよい。どちらか一方だけが与えられている場合は，ストリング定数が括弧でくくられていない限り，この因数はストリング繰返し因数であるとみなされる。ストリング繰返し因数は，無符号の１０進整数定数でなければならないことに注意されたい。たとえば，次の例について考えてみよう。

((2)'A')は('AA')に等しい。
((2)('A'))は('A','A')に等しい。
((2)(1)'A')は('A','A')に等しい。

12. 反復をネストすることができる。

13. 初期値のための式および関数参照の中で使用されている名前は，初期値化される項目が宣言されているブロックの中で知られているものでなければならない。

14. STATICラベルまたは入口変数にINITIAL属性を与えることはできない。

15. STATIC 属性をもたないステートメント・ラベル変数からなる配列の要素の初期値化には，別の方法を使用することができる。すなわち，ラベル配列の要素をステートメント接頭語として使用する方法である。ただし，この場合，添字はすべて１０進整数定数でなければならない（符号は任意選択）。このようにすると，配列のその要素は，添字付きの参照を接頭語としてもつステートメントのラベル定数の値に初期値化される。このステートメントは，配列の宣言を含んでいるブロックの直ぐ内部になければならない。１つのラベル配列の初期値化には，いずれか一方の方法しか使用することができない。この種の配列に対して CHECKが指定されており，しかも上述の方法でこの配列の要素が初期値化される場合，初期値化の際にはCHECK条件は引き起されない。

16. INITIAL 属性の中のある項目の属性が，その項目自体の属性と異なる場合，これらの属性が両立し得るものであるならば，変換が行なわれる。

17. ２つ以上の宣言の中であるSTATIC EXTERNAL項目にINITIAL属性を与える場合は，どの宣言においても値は同じでなければならない。

第２形のINITIAL属性に関する規則：

1. 受け渡される「入口式」および「引数のリスト」は，「第６章： プログラムの構成」で説明したプロローグに関する条件を満足するものでなければならない。

2. 第２形を使ってSTATICデータの初期設定を行なうことはできない。

例：

```
a. DECLARE SWITCH BIT (1)
           INITIAL ('1'B);

b. DECLARE MAXVALUE INITIAL (99),
           MINVALUE INITIAL (-99);

c. DECLARE A (100,10) INITIAL
           ((920)0,  (20)  ((3)5,9));

d. DECLARE TABLE (20,20) INITIAL
           CALL SET_UP  (X,Y);

e. DECLARE 1 A(8),
             2 B INITIAL (0),
             2 C INITIAL ((8)0);

f. DECLARE Z(3) LABEL;
           .
           .
           .
   Z(1):   IF X = Y THEN GO TO EXIT;
           .
           .
           .
   Z(2):   A = A + B + C * D;
           .
           .
           .
   Z(3):   A = A + 10;
           .
           .
           .
   GO TO Z(I);
           .
           .
           .
   EXIT:   RETURN;
```

例Cの結果は次のようになる。Aの最初の９２０個の要素は，各各０にセットされる。次の８０個の要素に関しては，５，５，５，９が２０回繰り返される。

例dの中のSET_UPは TABLE 内の諸要素の初期値をセットするためのプロシージャーの名前である。XとYは，SET_UPに渡される引数である。

例eにおいて，BとCは（８）と言う次元を引き継ぐ。ただし，Bに関しては最初の要素だけが初期値化されるのに対して，Cに関してはすべての要素が初期値化される。

最後の例において，Iに１，２，あるいは３という値を与えることによって，配列Ｚの特定の要素に転送が行なわれる。

INPUT, OUTPUT, UPDATE

INPUT属性，OUTPUT属性，およびUPDATE属性は，ファイルの機能を指示するのに使用される。INPUT属性は，補助記憶

装置からプログラムへデータを転送するよう指定するためのものである。OUTPUT属性は，新しいデータ・セットを作成するため，あるいは既存のデータ・セットを拡張するために，プログラムから補助記憶装置へデータを転送するよう指定するものである。UPDATEは，いずれの方向にもデータを転送できることを指定するものである。したがって，UPDATEファイルは，入力ファイルであると同時に出力ファイルでもある。

一般形式：

INPUT|OUTRUT|UPDATE

一般規則：

1. INPUT属性をもつファイルにPRINT属性を与えることはできない。

2. OUTPUT属性をもつファイルにBACKWARDS属性を与えることはできない。

3. UPDATE属性をもつファイルにSTREAM属性，BACKWARDS属性，あるいはPRINT属性を与えることはできない。SEQUENUIALファイルに対してUPDATE属性が宣言されている場合は，「その場で更新」方式がとられることを示している。この種のファイルにアクセスするときは，READの次にREWRITEがつづくという順序でなければならない。

省略時属性：

省略時属性はINPUTである。PRINT属性はOUTPUT属性を暗示する。

INTERNAL

EXTERNALの項を参照のこと。

IRREDUCIBLE，REDUCIBLE

省略形： IRREDUCIBLEに対してはIRRED
REDUCIBLEに対してはRED

これらは最適化に関する属性である。DOS PL／I 最適化コンパイラーによって処理されるプログラムの中にこれらの属性が出て来た場合は，その記入方法が構文法にかなっているか否かチェックされるだけで，これらの属性は無視される。このようなプログラムにこれらの属性が記入されていても，誤りではない。

KEYED

KEYED 属性は，当該ファイルの中のレコードをアクセスする際に，オプションKEY，KEYTO，およびKEYFROM が使用されることがあると指定するためのものである。これらのオプションは，ファイルの中のレコードにアクセスするのにキーが使用されることを示している。

一般形式：

KEYED

一般規則：

1. KEYEDファイルは，STREAM属性またはPRINT属性をもつことはできない。

2. KEYED属性は，RECORDファイルに対してだけ指定することができる。KEYED属性は，直接アクセス装置と結び付いていなければならない。

3. オプションKEY，KEYTO，およびKEYFROMのいずれかが使用されているファイルに対しては，必ず，KEYED属性を指定しなければならない。対応するデータ・セットに実際には記録されたキーが含まれている場合にも，上述のオプションを何も使用していなければ，KEYED属性を指定する必要はない。

省略時属性：

DIRECT属性はKEYED属性を暗示する。

LABEL

LABEL属性は，ここで宣言されている識別名はラベル変数であり，その値がステートメント・ラベルであることを指定するためのものである。目的プログラムの最適化の一助として，LABEL属性の指定の中に，当該プログラムを実行している間にこの名前がとり得る値を記入することもできる。

一般形式：

LABEL 〔（ステートメント・ラベル定数
〔，ステートメント・ラベル定数〕･･･）〕

一般規則：

1. ステートメント・ラベル定数のリストが与えられている場合は，GO TOステートメントまたはR－形式項目の中で使用されるときの変数の値は，そのリストのメンバーでなければならない。リストの中にあるラベル定数は，この宣言を含んでいるブロックの中で知られているものでなければならない。リストの中に記入することのできるラベル定数の数は，最高125である。

2. ラベル配列に対してLABEL属性を指定する場合には，括弧で囲んだステートメント・ラベル定数のリストを使用することができる。

3. PROCEDUREステートメントまたはENTRY ステートメントを識別するためにラベル変数を使用することはできない。また，ラベル変数に入口定数を割り当てることはできない。

4. ラベル変数がSTATIC属性をもっていない場合には，ラベル配列の要素を指定している添字付ラベルをステートメント・ラベル接頭語として指定することができる。しかし，ENDステートメントのENDというキーワードの後にこ

のようなラベルを記入することはできない。さらに詳しいことは，INITIAL属性の項を参照のこと。

5. STATICラベル変数にINITIAL属性を与えることはできない。

6. GO TOステートメントの中で使用されるラベル変数の値は，そのGO TO ステートメントが実行されるときに活用状態にあるブロックの中で用いられているラベル定数でなければならない。このようなラベル変数の値が有効でない場合は，チェックアウト・コンパイラーはERROR条件を引き起す。しかし，最適化コンパイラーのもとでは，このようなエラーが検出されるかどうかは保証できない。

7. ラベルとラベルを比較することができる。ラベルの比較演算で使用することのできる演算子は，＝と¬＝である。比較演算において，活用状態でないブロックで使用されているラベル定数を値とするラベル変数を指定しても，誤りにはならない。

8. ナル・ステートメントの接頭語として指定されているラベルとこのナル・ステートメントの直後のステートメントの接頭語として指定されているラベルとは比較しても等しくならない。たとえば，

```
A:;
B:    X = 7;
```

ラベルAはラベルBに等しくない。

9. FORMATステートメントの接頭語として指定されているラベルとこの後にあるステートメントの接頭語として指定されているラベルとは，比較しても等しくならない。

10. ENDステートメントの接頭語として指定されているラベルとこの後のステートメントの接頭語として指定されているラベルは，比較しても等しくならない。

11. IFステートメントのラベルはその後のTHEN 文節のラベルとは，比較しても等しくならない。

長さ属性

BITの項を参照のこと。

LIKE

LIKE属性は，ここで宣言されている名前は，LIKEの後に記入されている名前と同じ組立て方の構造変数であることを指定するためのものである。LIKE属性を与えると，小構造名，要素名，および，小構造名ならびに要素名の属性が同じになる。

一般形式：

LIKE　構造変数

一般規則：

1. 「構造変数」は大構造名でも小構造名でもよい。これは修飾された名前でもよいが，添字付の名前であってはならない。

2. 「構造変数」は，LIKE 属性の指定を含んでいるブロックの中で知られている名前でなければならない。LIKE属性が展開される前に，すべてのLIKE属性の中の構造名が，宣言されている構造と結び付けられる。たとえば，

```
DECLARE 1 A, 2 C, 3 E, 3 F,
        1 D, 2 C, 3 G, 3 H;
   .
   .
   .
BEGIN;
  DECLARE 1 A LIKE D ,1 B LIKE A.C;
  .
  .
  .
END;
```

この宣言の結果は，次のようになる。

1 A LIKE D は次のように展開される。

1 A, 2 C, 3 G, 3 H

1 B LIKE A.C は次のように展開される。

1 B, 3 E, 3 F

3. 「構造変数」もその小構造もLIKE 属性を使って宣言されたものであってはならない。また，「構造変数」を含んでいる構造がLIKE 属性によって完成させられたものであってはならない。

4. 作成された構造に小構造や要素名を追加することはできない。DECLAREステートメントでLIKE属性を指定するときに「構造変数」の直後に記入するレベル番号は，LIKE属性を使って宣言されている名前のレベル番号より代数的に小さいか等しくなければならない。

5. 「構造変数」自体の属性が新たに作成される構造に引き継がれるわけではない。たとえば，記憶域クラスの属性は引き継がれない。キーワードLIKE の後の「構造変数」が構造の配列を表わしている場合は，その次元属性は引き継がれない。

しかし，小構造名と要素名の属性は引き継がれる。また，含まれている次元属性と長さ属性は再計算される。ただし，ステートメントに接頭語をつけることによって初期設定されたラベル配列の要素に対するINITIAL 属性には，これは適用されない。

6. LIKE を使って宣言されている構造を記述にしたがって組み立てるとレベル番号の続き方が正確でなくなる場合は（たとえば，レベル3の小構造がレベル1の大構造とLIKE であると宣言されている場合），LIKE 属性を適用する前に定数によってレベル番号が修正される。

7. 構造に含まれている要素のALIGNED 属性およびUNALIGNED属性が引き継がれる前に，LIKE属性が展開される。

8. 標準省略時属性またはDEFAULT ステートメントが適用される前に，LIKE属性が展開される。

OFFSET, POINTER

省略形：　POINTERに対してPTR

OFFSET属性とPOINTER属性は，ロケーター変数について記述するためのものである。基底付変数のある特定の世代を参照するために，基底付変数参照の中でロケーター変数を使用することができる。オフセット変数は，エリアの最初の場所を基準にして，ある場所を識別するためのものである。ポインター変数は，任意の場所（エリアの中の場所を含む）を識別するためのものである。

一般形式：

```
POINTER|OFFSET
〔(要素エリア変数)〕
```

一般規則：

1. ポインター変数の宣言には，DECLARE ステートメントの中で明示的に宣言する方法，ポインター修飾語として指定することによって文脈的に宣言する方法，BASED属性の中で指定することによって文脈的に宣言する方法，SET オプションの中で指定することによって文脈的に宣言する方法がある。

2. オフセット変数を文脈的に宣言することはできない。エリア変数を指定してない場合は，オフセット変数の用途はPOINTER組込み関数を利用してロケーター修飾語として使用することだけである。

3. ポインター変数の値は，次のいずれかの方法でセットすることができる。

 a. READステートメントのSETオプションを利用する。

 b. LOCATEステートメントを利用する。

 c. ALLOCATEステートメントを利用する。

 d. 他のロケーター変数の値を割り当てる，または，ユーザー定義の関数から返されたロケーター値を割り当てる。

 e. ADDR，NULL，またはPOINTER組込み関数の値を割り当てる。

4. オフセット変数の値は，次のいずれかの方法でセットすることができる。

 a. ALLOCATEステートメントを利用する。

 b. 他のロケーター変数の値を割り当てる，または，ユーザー定義の関数から返されたロケーター値を割り当てる。

 c. NULLまたはOFFSET組込み関数の値を割り当てる。

5. 比較演算子＝および¬＝以外の演算子のオペランドとしてロケーター変数を使用することはできない。

6. ロケーター・データを他のデータ・タイプのデータに変換することはできない。しかし，ポインターをオフセットに変換したり，オフセットをポインターに変換したりすることはできる。

7. ロケーター値は，ロケーター変数にしか割り当てることができない。オフセット値がオフセット変数に割り当てられるとき，OFFSET属性の中で指名されているエリア変数は無視される。

8. STREAM入出力を用いてロケーター・データを転送することはできない。

9. ポインターとオフセットとの間で暗示的な変換が行なわれるときはいつでも，OFFSET属性の中で指定されているエリア変数を用いてその値が確定される。

省略時属性：

OFFSET属性の中で指定されている変数は，AREA 属性をもつものとして文脈的に宣言される。

OPTIONS

OPTIONS属性は，入口データの特性を指定するためのものである。OPTIONS属性は付加的属性であり，ENTRY 属性を暗示している。OPTIONS 属性は，渡される引数や総称で表わされている名前の選定には，何の効果も及ぼさない。

一般形式：

OPTIONS（オプション・リスト）

これは次のような方法で使用される。

DECLARE 識別名
〔ENTRY〔(パラメーター記述子リスト)〕〕
〔VARIABLE〕OPTIONS（オプション・リスト）；

オプションとオプションの間にはブランクを置く。当コンパイラーのオプションは次に示す通りである。

COBOL 〔追加オプション〕
FORTRAN〔追加オプション〕
{ASSEMBLER|ASM} INTER

COBOLまたはFORTRANの指定と共に追加オプションを1つ以上任意の順序で指定することができる。これらのオプションは次の通りである。

{ NOMAP 〔(引数リスト)〕
NOMAPIN 〔(引数リスト)〕
NOMAPOUT 〔(引数リスト)〕 }
INTER

ここでは，COBOL，FORTRAN，および追加オプションについて簡単に説明するにとどめる。これらの用途および働きについての詳細は，「第17章： 言語間の連絡機能」を参照されたい。

一般規則：

1. OPTIONS 属性は，入口宣言の中でしか使用することができない。また，外部入口定数ないし外部入口変数，およびパラメーターに対してしかOPTIONS属性を与えることができない。

2. オプションを記入する順序は任意である。

3. COBOLオプションは，指示されている入口点がCOBOLサブプログラムの中の入口点であることを指定するためのものである。

4. FORTRANオプションは，指示されている入口点がFORTRANルーチンの中の入口点であることを指定するためのものである。

5. ASSEMBLERオプションは，指示されている入口点がアセンブラー・ルーチンの中の入口点であることを指定するためのものである。このオプションは，以下の規則に従って，PL／Iプログラムから入口点を呼出すための便宜を提供する。

   a. OPTIONS(ASSEMBLER)属性が宣言されている入口名を，関数参照として使用することはできない。

   b. 入口点を呼び出すCALL ステートメントの中で，任意の数の引数（ゼロから入口宣言で暗示されている数まで）を渡すことができる。ただし，途中の引数を省略することはできない。

   c. INTERオプション（第7項の規則を参照のこと）が省略されている場合は，警告の診断メッセージが出され，INTERオプションが指定されているものとみなされる。ある種のPL／I規則に準拠していれば，アセンブラー・ルーチンが処理しない割込みはPL／I割込み取扱い機能によって処理される。（当コンパイラーのプログラマーの手引きを参照のこと）。

6. NOMAP，NOMAPIN，またはNOMAPOUTオプションを指定すると，PL／IとCOBOLないしFORTRANとの間のインターフェースにおいてデータ集合体の取扱いが行なわれなくなる。同一のOPTIONS属性の指定の中で，これらのオプションを1つまたはそれ以上記入することができる。

   オプション・キーワードの後にある任意選択の「引数のリスト」の中で，各オプションが適用される引数を指定することができる。「引数のリスト」の形式は以下の通りである。

   (ARGi〔,ARGj〕・・・)

   ここで，i，j，・・・は10進整数である。オプションはそれぞれ，プロシージャー参照の引数のリストにあるi番目，j番目・・・・・の項目に適用される。

   引数のリストの中には，オプションを適用すべき引数だけを記入する。引数を記入する順序は任意である。

   オプションの引数のリストが指定されていない場合は，当該入口名が呼び出されるときに渡されるすべての引数にそのオプションが適用される。

   ある1つのOPTIONS属性の指定の中で，指定または想定される2つ以上の引数のリストの中に同じ引数があってはならない。

7. INTERオプションは，PL／I以外のルーチンまたはサブプログラムの実行中に発生した割込みのうち，呼び出されたプログラムまたはその関連機能によって処理されない割込みを，PL／I割込み取扱い機能で処理するよう指定するためのものである。

   PL／Iによって割込みが正しく処理されるようにするためには，ASSEMBLER INTERオプションのもとで実行されるルーチンはPL／Iの規則に準拠しなければならない，という点に注意されたい。
   （当コンパイラーのプログラマーの手引を参照のこと)。

例：

```
DCL COBOLA OPTIONS(COBOL NOMAP(ARG1)
    NOMAPOUT(ARG3));
    .
    .
    .
CALL COBOLA(X,Y,Z);   /* X,Y,Z ARE
                         STRUCTURES */
    .
    .
    .;
DCL FRTRNA OPTIONS(FORTRAN INTER);
    .
    .
    .
CALL FRTRNA(L,M);     /* L AND M ARE
                               ARRAYS */
    .
    .
    .;
DCL ASSEMA OPTIONS(ASM INTER)
   ENTRY (FIXED DEC,,,FLOAT) ;
    .
    .
    .
CALL ASSEMA(A,B,C,D); /* VALID */
    .
    .
    .
CALL ASSEMA(A,B);     /* VALID */
    .
    .
    .
CALL ASSEMA;          /* VALID */
    .
    .
    .
CALL ASSEMA(A,,,D);   /* INVALID */
```

OUTPUT

INPUTの項を参照のこと。

パラメーター属性

パラメーター属性は，呼び出されるプロシージャーの中の名前が，そのプロシージャーへ渡される引数を表わしていることを指定するためのものである。

一般規則：

1. ある識別名をパラメーターのリストの中に記入すると，この識別名はパラメーター属性をもつものとして明示的に宣言される。この識別名に添字をつけたり，修飾したりしてはならない。

2. パラメーターのリストは，PROCEDUREまたはENTRYステートメントで指定される。パラメーター・リストの中のパラメーターは，左から右に向かって順に，引数リストの中の引数と対応している。引数の数とパラメーターの数は同じでなければならない。

3. 当該プロシージャーの内部にあるDECLARE ステートメントによって，パラメーター属性以外の属性を与えることができる。ただし，FILE以外のファイル属性をパラメーターに対して宣言することはできないし，STATIC属性，AUTOMATIC属性，BASED属性，BUILTIN属性，EXTERNAL属性，GENERIC属性，あるいはDEFINED 属性をパラメーターに対して宣言することもできない。

4. ストリング・オーバーレー定義の基底項目としてパラメーターを使用する場合，または，レコード方式の転送においてパラメーターを指定する場合には，明示的にCONNECTED属性を与えなければならない。

5. 星印または定数によって，単純パラメーターの限界，長さおよび大きさを指定しなければならない。INITIAL属性を与えることができるのは被制御パラメーターだけである。

6. ある引数の属性がこの引数に対応するパラメーターの属性と一致しない場合，パラメーターの属性に合致する属性をもつ見かけ引数が作り出される。この後，本来の引数が，変換され，見かけ引数に割り当てられる。内部入口定数に関しては，自動的にこの変換が行なわれる。外部入口定数および入口変数に関しては，変換が必要であるならば，該当する入口宣言においてパラメーター記述子のリストを与えなければならない。

   引数とパラメーターの関係については，「第9章： サブルーチンと関数」を参照されたい。

省略時属性：

DECLARE ステートメントで属性が与えられていない場合，パラメーター識別名の頭文字および対応するDEFAULTステートメントにもとづいて，省略時属性が適用される。パラメーターの省略時属性はINTERNALである。

PICTURE

省略形： PIC

PICTURE 属性は，文字ストリング・データおよび数字データの内部形式ならびに外部形式を定義し，データの編集方法を指定するのに使用される。数字データとは，算術値をもっているが，計算機内部では文字形式で貯えられるデータである。数字データは，算術演算が行なわれる前に，算術コードに変換される。

ピクチャー文字については，第II部の「ピクチャー指定用の文字」を参照されたい。

一般形式：

PICTURE

$\left\{ \begin{array}{l} \text{'文字ピクチャー指定　'} \\ \text{'数値ピクチャー指定　'} \end{array} \right\}$

「文字ピクチャー指定」も「数値ピクチャー指定」も，ピクチャー文字のストリングを単一引用符で囲んだものである。個々のピクチャー文字の前に繰返し因数を記入してもよい。繰返し因数は１０進整数定数，n，を括弧で囲んだもので，ピクチャー文字をn回繰返すことを表わす。nがゼロの場合，このピクチャー文字は無視される。ピクチャー文字の集まりはいくつかのフィールドに分けられるとみなされる。さらに，フィールドがいくつかのサブフィールドに分かれていることもある。

一般規則：

1. 「文字ピクチャー指定」は，文字ストリング・データ項目について記述するのに使用される。

2. 「数値ピクチャー指定」は，その用途に応じて，算術数値または文字ストリング値を表わす文字項目について記述するのに使用される。

3. 数字データ項目の基数は１０でなければならない。数字データ項目のスケールと精度は，ピクチャー文字によって指定する。基数属性，スケール属性，または精度属性と組み合わせてPICTURE属性を指定することはできない。ただし，数字データのモードがCOMPLEXである場合には，明示的にCOMPLEX属性を指定しなければならない。

4. 数値ピクチャー変数には，算術データまたは算術数値を表わす文字ストリング・データしか割り当てることができない。

POINTER

OFFSETの項を参照のこと。

POSITION

DEFINEDの項を参照のこと。

精度属性

精度属性は，当該データ項目の値を表わすのに必要な有効数字の最低桁数や，スケール因数（２進小数点ないし１０進小数点の仮想位置）を指定するのに使用される。精度属性は２進データにも１０進データにも適用される。

一般形式：

（桁数〔，スケール因数〕）

「桁数」は無符号の１０進整数定数であり，「スケール因数」は１０進整数定数（必要に応じて符号をつける）である。精度属性の指定は，（p，q）と略記されることもある。（pは「桁数」を表わし，qは「スケール因数」を表わす）。

一般規則：

1. 精度属性は，同じ表示レベルで因数化されているスケール属性（FIXEDまたはFLOAT），基数属性(DECIMALまたはBINARY)，あるいはモード属性（REALまたはCOMPLEX）の直ぐ後に記入しなければならない。この間にキーワードや名前があってはならない。

2. 桁数は，当該変数に割り当てられるデータ項目のために保持すべき桁数を指定するためのものである。スケール因数は小数部分の桁数を指定するためのものである。なお，スケール因数で指定された位置に実際に小数点が挿入されるわけではなく，その位置に小数点があると想定されるにすぎない。

3. スケール因数は，固定小数点変数に対してだけ指定することができる。桁数は，固定小数点変数に対しても浮動小数点変数に対しても指定することができる。

4. スケールがFIXEDで，スケール因数が指定されていない場合は，スケール因数はゼロであるとみなされる。すなわち，当該変数は整数を表わすとみなされる。

5. 変数のスケール因数，または，タイプがFIXEDである中間結果のスケール因数は，−１２８〜+１２７の範囲になければならない。

6. スケール因数は負でもよいし，桁数より大きくてもよい。負のスケール因数（−q）はつねに整数を表わす。このようなスケール因数を指定すると，実際の右端の桁よりq桁右に小数点があるものとみなされる。桁数より大きい正のスケール因数（q）はつねに小数部分を表わす。このようなスケール因数を指定すると，実際の右端の桁より９桁左に小数点があるものとみなされる。いずれの場合にも，小数点と実際の数字桁との間にゼロがあるとみなされるが，

このゼロは貯えられない。実際に貯えられるのは指定されている桁数だけである。

7. PICTURE属性と一緒に精度属性を指定することはできない。

8. 許容される最大の桁数は，１０進固定小数点データの場合は１５，２進固定小数点データの場合は３１，１０進浮動小数点データの場合は１６，2進浮動小数点データの場合は５３である。

省略時属性：

標準省略時精度は次に示す通りである。

DECIMAL	FIXED	(5,0)
BINARY	FIXED	(15,0)
DECIMAL	FLOAT	(6)
BINARY	FLOAT	(21)

PRINT

PRINT 属性は，当該ファイルのデータが最終的には印刷されるものであることを指定するものである。PUTステートメントのPAGE,LINE両オプションおよびOPEN ステートメントのPAGESIZEオプションは，PRINT属性をもつファイルに対してだけ使用することができる。これらのオプションについては，「J節： ステートメント」を参照されたい。

一般形式：

PRINT

一般規則：

1. PRINT属性は，OUTPUT属性およびSTREAM属性を暗示している。

2. PRINT属性とRECORD 属性とは両立しない。（ただし，RECORD ファイルを印刷装置と結び付けることはできる。「第１２章： レコード方式の転送」を参照のこと。）

3. PRINT属性を指定すると，ANS印刷装置制御文字を入れるために，各レコードの最初のデータ・バイトが確保される。この制御文字は，PAGE, SKIP, またはLINE形式項目ないしオプションによってセットされる。

省略時属性：

PUTステートメント でFILEもSTRINGも指定されていない場合には，標準出力ファイルSYSPRINTであるとみなされる。

REAL

COMPLEXの項を参照のこと。

RECORD, STREAM

RECORD属性とSTREAM 属性は，当該ファイルに対して用いられるデータ転送方式を指定するためのものである。STREAM属性は，当該ファイルのデータが，ストリームから変数へあるいは式からストリームへと割り当てられる文字形式の，連続するデータ項目の流れである，とみなされることを表わす。RECORD属性は，当該ファイルが物理的に分離しているレコードの集りからなっており，各々のレコードは任意の形式の１つ以上のデータ項目からなっていることを表わす。各レコードは，独立したものとして変数に（あるいは変数から）転送される。

一般形式：

RECORD|STREAM

一般規則：

1. STREAM属性をもつファイルは，OPEN,CLOSE,GET,PUT,ONの各ステートメントおよび割当てステートメントの中でだけ指定することができる。

2. RECORD属性をもつファイルは，OPEN,CLOSE,READ,LOCATE,WRITE,REWRITE,ON の各ステートメントおよび割当てステートメントの中でだけ指定することができる。

3. STREAM属性をもつファイルに以下の属性を与えることはできない。UPDATE,DIRECT,SEQUENTIAL,BACKWARDS,BUFFERED,UNBUFFERED,KEYED。 これらの属性はいずれもRECORD属性を暗示している。

4. RECORD属性をもつファイルにPRINT属性を与えることはできない。

省略時属性：

省略時属性はSTREAMである。

REDUCIBLE

IRREDUCIBLEの項を参照のこと。

RETURNS

RETURNS 属性は，入口変数ないし外部プロシージャーによ

って表わされる関数が返す値のデータ属性を定義するために，入口宣言の中で使用される。

一般形式：

RETURNS （属性・・・）

この属性は次のような方法で使用される。

DECLARE 識別名
〔ENTRY （パラメーター記述子のリスト）〕
〔VARIABLE〕 RETURNS （属性・・・）；

一般規則：

1. キーワードRETURNS の後に記入する括弧で囲んだリストの中の属性は，各々ブランクで区切らなければならない（ただし，精度属性のように括弧で囲む属性はこの限りでない）。これらの属性は，当該入口名が接頭語としてついているPROCEDUREないしENTRYステートメントのRETURNS オプションで明示的に指定されている属性または省略時属性と一致していなければならない。

2. これらの属性は，当該入口が関数として呼び出されたとき返される値のデータ特性を指定するためのものである。

3. ここで指定することのできる属性は，ストリング属性ないし算術属性（VARYINGを含む），または，ALIGNED，UNALIGNED，POINTER，OFFSET，AREA，FILE，EVENT，およびLABELだけである。OFFSET属性にはエリア名を記入することができるが，この場合，この名前は添字付の名前，修飾された名前，または被定義名であってはならない。また，AREA 属性にはエリアの大きさを記入することができるし，LABEL 属性にはラベル定数のリストを記入することができる。

4. 外部関数プロシージャーの，明示的に宣言されている入口定数に対してRETURNS 属性が指定されていない場合には，その入口定数の識別名にもとづいて省略時属性が適用される。標準省略時属性については以下で説明する。

   注： プロシージャー関数参照から返される値の属性は，RETURNS で指定されている属性と一致していなければならない。一致していないと，変換は行なわれないので誤りになる。

5. ストリングの長さおよびエリアの大きさは，10進整数定数で指定しなければならない。返される値は，ここで指定されている長さまたは大きさである。

省略時属性：

外部入口点に対してRETURNS 属性が指定されていない場合は，省略時属性をもつRETURNS 属性が指定されているものとみなされる。省略時属性は，DEFAULT ステートメントで指定されている属性または標準省略時属性である。IからNの間の文字で始まる入口の標準省略時属性はREAL FIXED BINARY(15,0)であり，これ以外の入口の標準省略時属性はREAL FLOAT DECIMAL(6)である。

SEQUENTIAL

DIRECTの項を参照のこと。

サイズ属性

AREAの項を参照のこと。

STATIC

AUTOMATICの項を参照のこと。

STREAM

RECORDの項を参照のこと。

UNALIGNED

ALIGNEDの項を参照のこと。

UNBUFFERED

BUFFEREDの項を参照のこと。

UPDATE

INPUTの項を参照のこと。

VARIABLE

VARIABLE属性は，ENTRY属性と一緒に使用したときはその

識別名が入口変数の名前であることを指定し，FILE 属性と一緒に使用したときはその識別名がファイル変数であることを指定し，LABEL 属性と一緒に使用したときはその識別名がラベル変数であることを指定するものである。

一般形式：

VARIABLE

一般規則：

1. VARIABLE 属性は，入口変数，ファイル変数，またはラベル変数の宣言の中でだけ指定することができる。

VARYING

BITの項を参照のこと。

J節： ステートメント

この節では，PL／Iの個々のステートメントについてアルファベット順に説明する。（プリプロセッサー・ステートメントは，この節の末尾でアルファベット順に取り上げる。）ステートメントの説明は，大抵の場合，次のような項目から構成されている。

1. ステートメントの意味と用法についての簡単な説明。
2. 一般形式 －－ ステートメントの構文法。
3. 構文法細則 －－ 一般形式の項で触れられていない構文法細則。
4. 一般規則 －－ PL／Iプログラムの中でのステートメントの用法および意味に関する規則。

ALLOCATE

ALLOCATEステートメントは，指定されている被制御データまたは基底付データに記憶域を割り振るためのものである。

一般形式：

ALLOCATEオプション〔，オプション〕・・・；

ここで、「オプション」は次のいずれかの形式をとる。

オプション1：
　「レベル」識別名「次元」
　　「属性」・・・

オプション2：
　基底付変数識別名
　　〔SET（要素ロケーター変数）〕
　　〔IN（要素エリア変数）〕

構文法細則1～6は，オプション1にだけ適用される。

1. 「レベル」のところにはレベル番号を記入する。キーワードALLOCATEの後の最初の識別名は，レベル1の識別名でなければならない。
2. 各識別名は，被制御記憶域クラスのデータを表わしているか，被制御大構造の要素でなければならない。
3. 「次元」のところには次元属性を記入する。「属性」のところにはBIT，CHARACTER，AREA，またはINITIAL を記入する。
4. 次元属性を記入する場合，その次元の数は対応する識別名の宣言で指定されている次元の数と等しくなければならない。
5. BIT属性は，BIT識別名に対してだけ与えることができる。CHARACTER属性はCHARACTER識別名に対してだけ与えることができる。AREA属性は，エリア識別名に対してだけ与えることができる。
6. ある要素を含んでいる大構造全体の相対的な組立て方が，この構造に対するDECLARE ステートメントの指定と同じである場合に限り，その構造メンバー名（大構造名は除く）を記入することができる。この場合，次元属性をもつものとして宣言されているすべての識別名に対してその次元属性を指定しなければならない。

構文法細則7～8は，オプション2にだけ適用される。

7. ALLOCATE ステートメントに記入する基底付変数は要素変数，配列，大構造のいずれであってもよい。これが大構造である場合，大構造名だけを指定する。
8. SETオプションを指定する場合，これはINオプションの前にあっても後にあってもよい。

一般規則：

一般規則1～6は，オプション1にだけ適用される。

1. オプション1を使用する場合に，ALLOCATE ステートメントで指定している識別名の記憶域がすでに割り振られたもので，まだ解放されていないならば，この識別名の記憶域は押し下げられる（すなわち，積み重ねられる）。ということは，この識別名の新しい世代のデータが作られるということである。この後，FREE ステートメントを使ってこの識別名の記憶域を解放すると，記憶域は押し上げられる（すなわち，積み重ねた記憶域が取り除かれる）。
2. 配列の限界，ストリングの長さ，およびエリアの大きさはALLOCATE ステートメントの実行のときに確定される。

   a. ALLOCATE ステートメントの中で限界，長さ，または大きさが明示的に指定されている場合には，その指定がとられ，DECLARE ステートメントの中での指定は無効になる。

b. ALLOCATE ステートメントの中で限界，長さ，または大きさとして星印を記入した場合は，現在の世代の限界，長さ，または大きさがとられる。変数の世代が存在しない場合は，限界，長さ，または大きさは定義できないので，そのプログラムはエラーになる。

c. ALLOCATE ステートメント，またはDECLARE ないしDEFAULT ステートメントの中で，識別名に与えるべき次元の大きさまたは長さ属性を指定しなければならない。DECLARE ないしDEFAULT ステートメントからとられた式の値は，割振りの時点で，ALLOCATE ステートメントで解禁された条件を用いて，計算される。ただし，識別名は，DECLARE ないしDEFAULT ステートメントの環境において解釈される。

d. ALLOCATE ステートメントまたはDECLARE ステートメントの中で，割り振りの行なわれる変数への参照を含んでいる式によって限界，長さ，またはエリアの大きさが指定されている場合は，その変数の一番新しい世代の値を使ってその式の値が計算される。

3. ある識別名がALLOCATE ステートメントまたはDECLARE ステートメントの中でINITIAL 属性を与えられている場合は，この識別名が割り振られるときにこれに初期値が割り当てられる。INITIAL 属性の中の式またはCALL オプションは，割振りのときに，ALLOCATE ステートメントで解禁された条件を使って，実行される。ただし，識別名はその宣言の環境において解釈される。DECLARE，ALLOCATE 両ステートメントでINITIAL 属性が指定されている場合は，ALLOCATE ステートメントのINITIAL 属性が使用される。初期値化を行なうのに今記憶域の割振りを行なっている変数を参照する必要がある場合，その変数の新しい世代が参照される。

4. ALLOCATION組込み関数を使用すると，ある識別名の記憶域が割り振られているか否か確かめたり，その識別名の世代がいくつあるか調べたりすることができる。

5. CONTROLLEDが宣言されているパラメーターを，ALLOCATE ステートメントの中で指定することができる。

6. ALLOCATE ステートメントの実行の時点で行なわれるいくつかの式の値の計算（たとえば，INITIAL 属性の中にあるいくつかの式の値の計算）は，互いに依存関係にあってはならない。

一般規則7～11はオプション2にだけ適用される。

7. オプション2を使用する場合は，記憶域は押し下げられない（つまり，積み重ねられない）。この場合は，ロケーター変数を用いて基底付変数のどの世代でも参照することができる。

8. 基底付変数の割り振りには割り振るべき基底付変数，新しい世代を識別するためのロケーター変数，および（エリアの中に新しい世代を割り振る場合には）エリアが関係する。SETオプションを指定しないと，SETオプションは基底付変数の宣言のBASED 属性で与えられているロケーター変数を指定しているものとみなされる。この場合，このBASED 属性でロケーター変数が指定されていなければ，誤りになる。SETオプションでオフセット変数を指定しており，INオプションを指定していない場合には，INオプションはオフセット変数の宣言のOFFSET 属性で与えられているエリアを指定しているものとみなされる。この場合，このOFFSET 属性でエリア変数が指定されていなければ，誤りになる。

9. SETオプションでオフセット変数を指定した場合は，新しい世代を表わすロケーター値がこのオフセット変数に割り当てられる。この場合，INオプションを指定するか，INオプションが想定されるようにしなければならない。また，このINオプションでは，オフセット変数の宣言のOFFSET 属性で指定されているエリアと同じものを指定するか，このエリアに含まれているかこのエリアを含むエリアを指定しなければならない。

10. INオプションが指定されておらず，しかもINオプションの指定が想定されない場合は，このALLOCATE ステートメントを実行するプログラムと結び付けられている記憶域に新しい世代が割り振られる。この場合，SETオプションでポインター変数を指定しなければならない。

11. ALLOCATE ステートメントでINオプションを指定している場合，またはINオプションの指定が想定される場合には，基底付変数の記憶域はここで指示されているエリアに割り振られる。このエリアに充分な大きさの記憶域がない場合は，AREA 条件が引き起される。

12. ある基底付変数に割り振られる記憶域の量は，割振りの時点でその変数に適用されている属性，次元の指定，長さの指定，および大きさの指定によって決まる。これらの属性は，基底付変数の宣言にもとづいて決定される。ALLOCATE ステートメントで属性を追加することはできない。基底付構造に，調節可能な配列の限界，ストリング

```
オプション1 （要素割当て）

  { 要素変数 } [ ，要素変数 ]
  { 擬似変数 } [ ，擬似変数 ] ・・・ ＝ 要素式；

オプション2 （配列割当て）

                                    { 構造式 〔，BY NAME〕 }
  {配列変数} 〔，配列変数〕・・・ ＝{ 配列式 〔，BY NAME〕 }；
                                    { 要素式                }

オプション3 （構造割当て）

  構造変数 ，構造変数          { 構造式 〔，BY NAME〕 }
  擬似変数 ，擬似変数  ・・・ ＝{ 要素式                }；
```

図J.1　割当てステートメントの一般形式

の長さ，またはエリアの大きさが含まれていてもよい（第I部，「記憶域の制御」の「REFER オプション」を参照のこと。）基底付変数の限界，長さ，大きさを指定するのに星印を使用することはできない点に注意されたい。

割当てステートメント

割当てステートメントは，式の値を計算し，その値を1つ以上の目標変数に割り当てるのに使用される。目標変数は要素変数，配列変数，構造変数のいずれであってもよい。また，目標変数が擬似変数であってもよい。

一般形式：

割当てステートメントには，3つの一般形式がある。これを図J.1に示す。

構文法細則：

1. オプション2では，各々の目標変数は配列でなければならない。割当て符号の右側に構造の配列がある場合，目標変数もすべて構造の配列でなければならない。割当て符号の右側に少なくとも1つの構造がある場合にだけ，BY NAME オプションを与えることができる。

2. オプション3では，各々の目標変数は構造でなければならない。

一般規則：

1. 集合体割当て（オプション2と3）は，一般規則5～8に従って，一連の要素割当てに展開される。

2. 要素割当ては次のようにして行なわれる。

   a. 目標変数の添字とロケーター修飾語の値，および，SUBSTR 擬似変数参照の2番目と3番目の引数の値が計算される（添字の値が計算される順序は不定である。）

   b. 次いで，割当て符号の右側にある式の値が計算される。

   c. 各目標変数ごとに（左から右に向かって），データ変換規則に従って，式が目標変数の特性に変換される（ただし，算術データの基数の変換が行なわれる場合はつねに，値が，直接，目標変数の精度に変換される）。この後，変換された値が目標変数に割り当てられる。

3. 以下の規則が，ストリング要素割当てに適用される。

   a. 割当ては，左端の桁から始まり右に向かって進められる。

   b. 目標変数が固定長ストリングである場合，式の値が長すぎるならばその右の方が切り捨てられる（切捨てが行なわれると，STRINGSIZE条件が解禁状態にあればこの条件が引き起される）。また，式の値が短かすぎるな

らば，右の方にパディングが行なわれる（文字ストリングの場合には，ブランクのパディングが行なわれ，ビット・ストリングの場合にはゼロのパディングが行なわれる）。（ストリング擬似変数は，固定長ストリングであるとみなされる点に注意されたい）。こうして得られた値が目標変数に割当てられる。

c．目標変数がVARYING ストリングであり，この変数の宣言で指定されている最大の長さよりも式の値が長い場合には，値の右の方が切り捨てられる(STRINGSIZE条件が解禁状態にあれば，この条件が引き起される）。目標ストリングの現時点の長さは，その最大の長さになる。式の値が最大の長さより長くない場合には，その値が割り当てられる。この場合，目標ストリングの現時点の長さは値の長さに等しくなる。

4. その他の要素割当てには，以下の規則が適用される。

a．目標変数がエリア変数である場合には，式はエリア変数か関数でなければならない。割り当てられるエリアの現時点の範囲にくらべて目標エリアの大きさが不充分である場合は，この割当てによってAREA 条件が引き起される。

b．目標変数がポインター変数である場合には，式はポインター変数（ないしオフセット変数）またはポインター関数参照（ないしオフセット関数参照）でなければならない。式がオフセット・タイプである場合は，その値がポインターに変換される。

c．目標変数がオフセット変数である場合には，式はオフセット変数（ないしポインター変数）またはオフセット関数参照（ないしポインター関数参照）でなければならない。式がポインター・タイプである場合は，その値がオフセットに変換される。

d．目標変数がラベル変数である場合には，式はラベル式でなければならない。環境情報（ブロックの呼出しを識別するための情報）は，つねに，ラベル変数に割り当てられる。

e．目標変数が事象変数である場合には，式も事象変数でなければならない。この割当ての間に割込みが起こることはなく，完了値と状況値の双方が割り当てられる。事象変数は，活用状態にある事象変数が割り当てられている間に，活用状態になることはない。活用状態にある事象変数に割当てを行なうと誤りになる。

f．目標変数がSTATUS 擬似変数である場合には，事象変数が活用状態にあるか否かにかかわりなく，値を割り当てることができる。名前を指定した事象変数が活用状態にあるときにCOMPLETION擬似変数に割り当てを行なうと誤りになる。

g．目標変数が入口変数である場合には，式は入口式でなければならない。

h．目標変数がファイル変数である場合には，式はファイル式でなければならない。

5. 集合体割当ての最初の目標変数は，マスター変数と呼ばれる。マスター変数が配列である場合には，配列展開（規則6）が行なわれる。これ以外の場合には，構造展開（規則7，8）が行なわれる。目標変数への割当てに関するCHECK 条件が解禁状態にあれば，各要素への割当ての後CHECK 条件が引き起される。BY NAME 割当ての場合には，ある項目に値が割り当てられたか否かにかかわりなく，目標変数に関してCHECK 条件が引き起される。もとのステートメントのラベル接頭語は，生成された他のステートメントの前にあるナル・ステートメントに適用される。

6. オプション2では，すべての配列オペランドの次元数および限界が同じでなければならない。配列割当ては次のような形式のループに展開される。

```
LABEL: DO j1 = LBOUND(マスター変数，1) TO
               HBOUND(マスター変数，1);

       DO j2 = LBOUND(マスター変数，2) TO
               HBOUND(マスター変数，2);
         .
         .
         .
       DO jn = LBOUND(マスター変数，n) TO
               HBOUND(マスター変数，n);

          生成された割当てステートメント

       END LABEL;
```

この展開において，n は割当てに関与するマスター変数の次元の数である。生成された割当てステートメントにおいては配列オペランドにはすべて，（左から右に向かって順に）見かけ変数 j1 — jn を用いて，完全な形の添字がつけられる。配列オペランドに添字がついていないときは，この配列オペランドは j1 — jn の添字だけをもつことになる。

横断面表記が使用されている場合には，星印がj1 — jn に置き換えられる。もとの割当てステートメント（これが規則7または規則8に準じて生成されたステートメントであることもある）が条件接頭語をもつ場合には，生成された割当てステートメントにもこの条件接頭語がつけられる。もとの割当てステートメント（これが規則8に準じて生成されたものであることもある）がBY NAME オプションをもつ場合には，生成された割当てステートメントにもBY NAME オプションが与えられる。生成された割当てステートメントが構造割当てである場合には，このステートメントは以下のようにして展開される。

7. オプション3においてBY NAME オプションが指定されていない場合には，次のような規則が適用される。

   a. どのオペランドも配列であってはならない。ただし，オペランドが，配列を含んでいる構造であってもよい。

   b. 構造オペランドに直接含まれている項目の数（k）は，すべての構造オペランドに関して同じでなければならない。

   c. 割当てステートメント（これが規則6に準じて生成されたステートメントであることもある）は，k個の生成された割当てステートメントに置き換えられる。生成された割当てステートメントのi番目のものは，もとの割当てステートメントにおいて各々の構造オペランドをその中に含まれているi番目の項目で置き換えることによって得られた割当てステートメントである。このようにして生成された割当てステートメントを，さらに，規則6または規則7に従って，展開しなければならないこともある。生成されたすべての割当てステートメントには，もとのステートメントの条件接頭語がつけられる。

8. オプション3においてBY NAME オプションが与えられている場合には，構造割当て（これが規則6に準じて生成された割当てであることもある）は，以下のa～dに従って展開される。この場合，どのオペランドも配列であってはならない。

   a. マスター変数に直接含まれている最初の項目が検討される。

   b. 各構造オペランドおよび目標変数に直接含まれている項目の中に同じ識別名をもつものがある場合には，次のようにして割当てステートメントが生成される。各々の構造オペランドおよび目標変数を，各々に直接含まれておりしかもこの識別名をもつ項目によって置き換えることにより，割当てステートメントが得られる。このような識別名を含んでいない構造があると，割当てステートメントは生成されない。生成された割当てが構造割当てまたは構造の配列割当てである場合には，BY NAME がつけられる。最初に生成された割当てには，もとの割当てステートメントのラベル接頭語が与えられる。また，生成されたすべての割当てステートメントにもとの割当てステートメントの条件接頭語が与えられる。

   c. マスター変数に直接含まれている各々の項目について，bの操作が繰り返される。マスター変数に含まれている項目の順に，割当てステートメントが生成される。

   d. aからcの操作によって，さらに，配列割当てや構造割当てが生成されることがある。これらの割当ては，規則6～8に従って，展開される。

BEGIN

BEGIN ステートメントは，開始ブロックの冒頭にあって開始ブロックを識別するためのものである。

一般形式：

BEGIN〔ORDER|REORDER〕;

構文法細則：

1. BEGIN ステートメントのラベルは添字付きのものでもよい。しかし，END ステートメントにこのようなラベルを記入することはできない。

一般規則：

1. BEGIN ステートメントは，ENDステートメントと対になるもので，ある1つの開始ブロックを区切るのに使用される。開始ブロックについての詳細は，「第6章： プログラムの構成」を参照のこと。

2. ORDERとREORDERは最適化に関するオプションである。すなわち，ORDERとREORDERは，どの程度まで当該ブロックを最適化すべきか指定するためのものである。一般に，ORDER を指定すると，あるブロックにおいてセットされた変数の最も新しい値は，そのブロックの実行中に任意の場所から入った計算関係のオン・ユニットの中で使用することができる，という程度まで最適化が進められる。REORDER を指定すると，これ以上に最適化が進められる。したがって，REORDER が指定されていると，あるブロックの実行中に入ったオン・ユニットにおいて，そのブロックでセットされた変数の最も新しい値が使用できないこともある。

CALL

CALL ステートメントは，プロシージャーを呼び出し，そのプロシージャーの中の指定されている入口点に制御を渡すよう指示するためのものである。

一般形式：

```
CALL {入口式|総称名}
  〔(引数〔,引数〕・・・)〕
```

構文法細則：

1. 入口式　総称名ないし組込み名によって，呼び出されるサブルーチンの入口点を指示する。

2. 入口式が表わしている入口定数が，外部入口定数であり，当該CALLステートメントを含んでいるブロックの中で知られていないものであると，誤りになる。したがって，当該ブロックの外部にあるこのような入口定数を使用する場合には，当該ブロックにおいてこれらすべてを明示的に宣言しなければならない。

一般規則：

1. RETURNステートメント(RETURN式)によって呼出しプロシージャーに制御が戻される場合には，プロシージャーの呼出しにCALL ステートメントを使用してはならない。

2. 引数と，呼び出されたプロシージャーの中でこの引数を表わすパラメーターとが相互にどのように影響を及ぼし合うかについては，「第9章：　サブルーチンと関数」を参照のこと。

CLOSE

CLOSE ステートメントは，ここで指定されているファイル式と当該ファイルが開かれたときにこのファイル式に結び付けられたデータ・セットとの関係を断ち切るためのものである。

一般形式：

```
CLOSE FILE(ファイル式)
 〔ENVIRONMENT(LEAVE|UNLOAD)〕
 〔,FILE (ファイル式)
 〔ENVIRONMENT(LEAVE|UNLOAD)〕〕・・・;
```

一般規則：

1. FILEオプションは，どのファイルを閉じるかを指定するためのものである。1つのCLOSEステートメントで2つ以上のファイルを閉じることができる。ただし，各ファイルに対してFILEオプションを1つずつ指定しなければならない。

2. 閉じられたファイルを再び開くことができる。

3. 開かれていないファイルを閉じたり，すでに閉じられているファイルを再び閉じるよう指定しても，何も行なわれない。

4. CLOSEステートメントによって閉じられていないファイルがあると，当該プログラムが完了したとき自動的にこのファイルが閉じられる。

5. 当該ファイルと結び付けられている入出力事象のうち当該ファイルが閉じられるとき状況値が0であるものは，すべて，状況値1にセットされる。

6. LEAVEオプションまたはUNLOADオプションをつけて，ENVIRONMENT 属性を与えることができる。

DECLARE

省略形：　DCL

DECLARE ステートメントは，名前の属性を明示的に宣言するための主要な手段である。

一般形式：

```
DECLARE
 〔レベル〕識別名〔属性〕・・・
 〔SYSTEM〕
 〔,〔レベル〕識別名〔属性〕・・・
 〔SYSTEM〕〕・・・;
```

構文法細則：

1. 1つのDECLARE ステートメントの中で，識別名をいくつでも宣言することができる。

2. 「レベル」は，ゼロ以外の無符号の10進整数定数である。レベル番号を指定しないと，要素変数および配列変数はすべてレベル1であるとみなされる。大構造名を宣言する場合には、つねにレベル1を指定しなければならない。なお，レベル番号とその後の識別名との間にはブランクを1つ置かなければならない。

3. DECLARE ステートメントで指定する属性と属性の間には，ブランクを置かなければならない。次元属性，長さ属性，および精度属性以外は，任意の順序で属性を記入することができる。次元属性は，配列名の直ぐ後に書かなければならない。長さ属性と精度属性は，これらと結び付いているいずれかの属性の後になければならない。ある特定の名前に対して指定されている最後の属性の後には，コンマを記入しなければならない(属性を1つも指定しない場合には，名前の後にコンマを記入しなければならない)。ただし，DECLARE ステートメントの最後の名前の最後の属性の後には，コンマではなくセミコロンを記入する。

4. 「SYSTEM」は，対応する識別名に標準省略時属性を適用するよう指定するためのものである。

この場合，DEFAULT ステートメントから属性がとられるわけではない。「SYSTEM」は，他の属性の前，後，間のどこにあってもよい。

共通属性の因数化

宣言の中にあるいくつかの名前に共通な属性を因数化してくくり出すことができる。このようにすると，多くの識別名に対して同じ属性を繰り返し指定する必要がなくなる。共通属性の因数化とは，具体的には，名前を括弧で囲んで，この後にこれらの名前に適用される属性を記入することである。因数化された属性はすべて，括弧で囲まれているすべての名前に適用されるものでなければならない。因数化された属性をある特定の名前に関してだけ取り消すことはできない。ただし，括弧で囲んであるどの名前にも（因数化されている属性と両立し得る）他の属性を与えることができる。共通属性の因数化は，DECLARE，DEFAULT 両ステートメントでだけ許されるものであり，ENTRY属性の宣言においてこれを行なうことはできない。次元属性を因数化することもできる。精度属性および長さ属性に関しては，これらの属性のキーワード属性といっしょにこれらの属性を因数化する場合に限り，因数化が認められる。次の４番目の例に示すように，共通属性の因数化をネストすることができる。

括弧の中の名前と名前の間には，コンマを記入する。

注： 構造のレベル番号を因数化することもできる。ただし，この場合，括弧で囲まれた名前の前にレベル番号を記入しなければならない。

```
DECLARE(A,B,C,D) BINARY FIXED (31);

DECLARE(E DECIMAL(6,5),
        F CHARACTER(10)) STATIC;

DECLARE 1 A, 2(B,C,D) (3,2) BINARY
        FIXED (15),・・・;

DECLARE ((A,B) FIXED(10), C FLOAT(5))
         EXTERNAL;
```

一般規則：

1. レベルの１識別名は，特定のブロック内の１つのDECLARE ステートメントの中でだけ指定しなければならない。したがって，その識別名に明示的に与える必要のあるすべての属性をそのDECLARE ステートメントの中で，宣言しなければならない。（ただし，FILE属性をもつ識別名にはOPEN ステートメントの中でも属性を与えることができる。詳しくは，この節の「OPEN ステートメント」および「第１０章：入力と出力」を参照のこと。）

2. いろいろなブロックおよびコンパイレーションにおける外部名の属性は，首尾一貫していなければならない（INITIAL属性は，１度与えるだけでよいから，この適用外にある）。

3. DECLARE ステートメントにラベル接頭語をつけることができる。ただし，このようなラベルへのブランチは，ナル・ステートメントへのブランチとみなされる。DECLARE ステートメントに条件接頭語をつけることはできない。

DEFAULT

省略形： DFT

DEFAULT ステートメントを利用すると，プログラマーの側で，属性の一部またはすべてを暗示的に宣言する必要のある識別名に適用される省略時属性を指定することができる。DEFAULT ステートメントを使用すると，次の１～４の省略時属性を指定することができる。

1. 明示的に宣言されている識別名。
2. 文脈的に宣言されている識別名。
3. パラメーター記述子の中に入れる属性。
4. 暗示的に宣言されている識別名と関数プロシージャーから返される値。

一般形式： 図J.2を参照のこと。

一般規則：

1. 不完全なところのある明示宣言ないし文脈宣言，および，暗示宣言に関するDEFAULT ステートメントの規則にしたがつて適用されない属性は，標準省略時属性の適用規則に準じて与えられる。

2. DEFAULT ステートメントの有効範囲は，このステートメントが入っているブロック，ならびに，そのブロック内にあって，同じ範囲の別のDEFAULT ステートメントを含まずまた同じ範囲のDEFAULT ステートメントをもつブロックにも含まれていない，すべてのブロックである。

   外側のブロックのDEFAULT ステートメントの範囲が，その中に含まれているブロックの中のDEFAULT ステートメントの範囲と部分的に重複することも可能である。このような場合には，外側のブロックのDEFAULT ステートメントの範囲が，含まれているブロックの中のDEFAULT ステートメントの範囲だけ小さくなる。たとえば，

```
 P:  PROCEDURE;
L1:  DEFAULT RANGE (XY) FIXED;
      .
      .
      .
 Q:  BEGIN;
L2:  DEFAULT RANGE (XYZ) FLOAT;
 END P;
```

```
DEFAULT   {単純指定|因数化された指定}
          〔,{単純指定|因数化された指定}〕・・・;
「単純指定」は,
 ⎧ RANGE  ({識別名|文字:文字}
 ⎪        〔,{識別名|文字:文字}〕・・・)〔属性の指定〕 ⎫
 ⎨ RANGE(*)〔属性の指定〕                                ⎬
 ⎩ DESCRIPTORS〔属性の指定〕                             ⎭
「因数化された指定」は,
    ({単純指定|因数化された指定}
   〔,{単純指定|因数化された指定}〕・・・)〔属性の指定〕
「属性の指定」は,
 ⎧ 属    性  ・・・〔VALUE  (値の指定)〕 ⎫
 ⎩ VALUE  (値の指定)                      ⎭
「値の指定」については本文を参照のこと。
```

図J.2 DEFAULT ステートメントの一般形式

L1というDEFAULT ステートメントの有効範囲は,プロシージャーPの中にあって文字XYで始まっているすべての識別名,および,開始ブロックQの中にあって文字XYで始まっている識別名のうち文字XYZで始まっている識別名を除いたすべての識別名,である。L2と言うDEFAULTステートメントの範囲および有効範囲は,開始ブロックQの中にあって文字XYZで始まっているすべての識別名である。

3. VALUE (値の指定)は,属性の指定の中で配列の次元属性の前以外の任意の場所に記入することができる。

4. VALUEは,ストリングの長さ,エリアの大きさ,および精度に関する省略時規定を設定するためのものである。特定の属性をもつ特定の精度指定を指示するために,「値の指定」の中に基数属性および/またはスケール属性を記入しなければならない。

5. 「値の指定」は,以下の指定を1つ以上任意の順序で並べたものである。

   a. AREA(大きさ)

   b. BIT (長さ)

   c. CHARACTER (長さ)

   d. {基数属性スケール属性|
      スケール属性基数属性}
      (精度〔,スケール因数〕)

展開されたとき上のような形式になるのであれば,共通の基数属性とスケール属性を因数化することができる。

BITデータまたはCHARACTER データの長さは,式でも10進整数定数でもよい。

例:

```
DEFAULT RANGE(A:C)
        VALUE (FIXED DECIMAL(10),
               FLOAT DECIMAL(14),
               AREA(2000));
DECLARE B FIXED DECIMAL,
        C FLOAT DECIMAL,
        A AREA;
```

これらのステートメントは次のステートメントに等しい。

```
DECLARE B FIXED DECIMAL(10),
        C FLOAT DECIMAL(14),
        A AREA(2000);
```

6. RANGEは,DEFAULTステートメントの中で指定されている属性が適用される識別名を指定するためのものである。

   a. 「RANGE 識別名」という形は,「識別名」で示されている文字(1字以上)で始まっている識別名に省略時規定を適用するよう指定するためのものである。たとえば,

      RANGE(ABC)

   は,次のような識別名に適用される。

      ABC

ABCD
ABCD・・・・など

しかし，次のような識別名には適用されない。

ABD
ACB
AB
A

RANGE の後で1字だけ指定すると，この文字を頭文字とするすべての識別名に適用される。

b. RANGE の指定には，このほかに，「文字：文字」という形がある。この形は，指定されている2つの文字のいずれかまたはアルファベットにおいてこの2つの文字の間にある任意の文字を頭文字とする識別名に，ある特定の範囲に対して指定されている省略時属性を適用するよう指定するためのものである。ここに記入する文字はアルファベットの順になっていなければならない。たとえば，

RANGE(A:G,I:M,T:Z)

c. RANGE(＊)という指定は，当該DEFAULT ステートメントの有効範囲にあるすべての識別名を指定するものである。

7. DESCRIPTORSは，明示的な入口宣言のパラメーター記述子のリストにあるパラメーター記述子に，このキーワードの後に記入されている属性を入れるよう指定するためのものである。ただし，記述子の中に同一クラスの択一属性があることによってこのような属性の組入れが禁止されていないこと，および少くとも1つの属性がすでに入っていることが条件である。この2番目の制約のために，DESCRIPTORS 省略時属性は，ナル記述子をもつパラメーター（対応する引数と同じ属性をもつパラメーター）には適用されない。

8. 「因数化された指定」： これらは次のようにして使用される。

```
DEFAULT (RANGE(A)FIXED, RANGE(B)
         FLOAT)BINARY;
```

このステートメントの結果，頭文字がAであり暗示的に宣言されている識別名の省略時属性はFIXED BINARYになり，頭文字がBであり暗示的に宣言されている識別名の省略時属性はFLOAT BINARYになる。

9. DEFAULT ステートメントにラベルをつけることができる。ただし，このラベルへのブランチはナル・ステートメントへのブランチであるとみなされる。DEFAULT ステートメントに条件接頭語をつけることはできない。

DEFAULT ステートメントで指定する属性に関する規則：

1. ENTRY 属性，ENVIRONMENT属性，RETURNS 属性，LIKE属性，VARIABLE 属性，およびFILE以外のファイルに関する属性を「属性の指定」で指定することはできない。FILEを指定した場合には，INTERNAL属性とVARIABLE 属性が暗示される。

2. DEFAULT ステートメントを用いて構造を作成することはできない。構造の要素には，その要素の識別名にもとづいて省略時属性が適用される（修飾された構造要素名にもとづいて省略時属性が適用されるわけではない）。

3. 次の属性に関しては，以下で記述されている各々に関する制約を守れば，「属性の指定」に記入することができる。

AREA — 大きさの指定なし

BIT — ストリングの長さの指定なし

CHARACTER — ストリングの長さの指定なし

LABEL — ラベルのリストなし

算術データの基数属性とスケール属性
— 精度の指定なし

4. パラメーターまたはパラメーター記述子にCONTROLLED属性を適用することはできない。パラメーター名である識別名に関しては，省略時属性として指定されている。CONTROLLEDは無視される。また，DESCRIPTORS の後の「属性の指定」にCONTROLLEDを指定しても，無視される。

5. 配列の次元を属性として指定することができるが，その場合は，「属性の指定」の中の最初の項目として指定しなければならない。次元の指定は，式であっても，10進整数定数であってもよい。
たとえば，

```
DEFAULT RANGE (J) (5);
DEFAULT RANGE (J) (5,5) FIXED;
```

しかし，次のように指定することはできない。

```
DEFAULT RANGE (J) FIXED (5);
```

6. INITIAL 属性を指定することができる。

DELAY

DELAY ステートメントは，指定されている時間だけプログラムの実行を停止するためのものである。

一般形式：

DELAY （要素式）；

一般規則：

1. DELAY ステートメントが実行されると，ここで指定されている要素式の値が計算され，その値が整数（n）に変換される。このnはミリセカンドを表わす。当コンパイラーにおいては，nは一番近い秒単位の整数にまるめられる。こうして得られた秒数だけ実行が停止される。

2. 計時機構が使用可能でない場合には，DELAY ステートメントはナル・ステートメントとして働く。

DELETE

DELETE ステートメントは，UPDATE ファイルからレコードを削除するためのものである。

一般形式：

DELETE FILE （ファイル式）
〔KEY（式）〕〔EVENT （事象変数）〕；

一般規則：

1. オプションを記入する順序は任意である。

2. FILE オプションは，UPDATE ファイルを指定するためのものである。必ずFILE オプションを指定しなければならない。

3. 当該ファイルがDIRECT UPDATE ファイルである場合には，KEYオプションを指定しなければならない。当該ファイルがINDEXED 編成（または，キー順VSAM 編成）のSEQUENTIAL UPDATE ファイルである場合にも，KEYオプションを指定することができる。ここで指定されている式は，文字ストリングに変換される。この文字ストリングにもとづいて，削除されるレコードが確定される。

4. 当該ファイルがSEQUENTIAL UPDATE ファイルである場合には，最後に読み取られたレコードが削除される。この場合，データ・セットの編成は，INDEXED （または，キー順VSAM）でなければならない。

5. EVENT オプションを指定すると，レコードの削除が行なわれているときにも処理を続行することができる。このオプションをもつDELETE ステートメントに制御が渡ると，「事象変数」が活用状態になり（つまり，この変数を他の事象と結び付けることができなくなる），暗示的なファイルの開放によってUNDEFINEDFILE 条件が引き起されない限り（下の「注」を参照のこと）この「事象変数」に'0'Bという完了値が与えられる。この「事象変数」を指定しているWAIT ステートメントに制御が渡るまでは，この「事象変数」は活用状態にとどまり，'0'Bと言う完了値を保持している。この「事象変数」を指定しているWAIT ステートメントに制御が渡ると，次のいずれかの措置がとられる。

   a. DELETE ステートメントが正しく実行され，そのDELETE の結果としてTRANSMIT 条件もKEY条件も引き起されなかった場合には，「事象変数」が「完了」にセットされ（すなわち，この変数に'1'Bと言う完了値が与えられ），この変数の状態は非活用状態になる（すなわち，この変数を他の事象と結び付けることができるようになる）。

   b. DELETE ステートメントを実行した結果TRANSMIT条件ないしKEY条件が引き起された場合には，上述のWAIT ステートメントが出て来るまではこれら各々の条件のための割込みは起らない。上述のWAIT ステートメントが出て来ると，条件が引き起された順に，各々に対応するオン・ユニットに入る（もちろん，オン・ユニットが指定されていなければこの限りでない）。最後のオン・ユニットから戻ったとき，あるいは，いずれかのオン・ユニットがGO TO ステートメントによって中止されたときに，「事象変数」に'1'B という完了値が与えられ，「事象変数」は非活用状態をとる。

   注： DELETE ステートメントによって暗示的にファイルが開放され，この結果UNDEFINEDFILE 条件が引き起された場合には，直ちに，この条件のためのオン・ユニットに入る。ただし，「事象変数」は何も変化しない。つまり，「事象変数」は依然非活用状態にあり，その値はDELETE ステートメントが出て来たときにこの変数がもっていた値と同じである。オン・ユニットの中でUNDEFINEDFILE 条件が訂正されなかった場合には，オン・ユニットからの正常な戻りの後，ERROR 条件が引き起される。オン・ユニットの中でこの条件が訂正された場合（すなわち，ファイルが正しく開かれた場合）には，オン・ユニットからの正常な戻りの後，事象変数が'0'Bにセットされ，この変数は活用状態となり，DELETE ステートメントの実行が続行される。

6. DELETE ステートメントがレコードをアンロックするのは，DELETE ステートメントがあるタスクの中でそのレコードがロックされた場合だけである。

7. DELETE ステートメントによって，ファイルが暗示的に開放されることがある。

例：

```
DELETE FILE(ALPHA) KEY (DKEY);
```

このステートメントが実行されると，ALPHA と言うファイルと結び付いているデータ・セットから，DKEY によって識別されるレコードが削除される。

DISPLAY

DISPLAY ステートメントは操作員にメッセージを表示するためのものである。応答を要求することもできる。

一般形式：

オプション1：

```
DISPLAY （要素式）；
```

オプション2：

```
DISPLAY （要素式）
REPLY   （文字変数|擬似変数）
〔EVENT （事象変数）〕；
```

一般規則：

1. DISPLAY ステートメントが実行されると，要素式の値が計算され，必要に応じて，要素式が長さの変わる文字ストリングに変換される。当コンパイラーにおいては，この文字ストリングの最大の長さは１２６桁である。この文字ストリングがメッセージとして表示される。

2. オプション2においては，文字変数または擬似変数が操作員から与えられたメッセージ（ストリング）を受け取る。STRING擬似変数を使用することはできない。また，メッセージは７２桁を越えてはならない。

3. オプション2においてEVENTオプションを指定しない場合には，DISPLAYとREPLYの双方に対して同じ変数を指定してはならない。

4. オプション2においてEVENTオプションが指定されていない場合には，操作員のメッセージを受け取るまでプログラムの実行が止まる。オプション1においては，実行は中断されることなく進行する。

5. EVENT（事象変数）オプションが与えられている場合には，応答が完了するのを待たずに次のステートメントの実行が始まる。応答が完了するまでは事象変数の完了値は'0'Bであり，応答が完了するとこの完了値が'1'Bになる。この事象を指定しているWAITステートメントが実行されて初めて，応答が完了したものとみなされる。先の応答が完了するまでは，他のDISPLAY ステートメントを出してはならない。

例：

```
DISPLAY ('END OF JOB');
```

このステートメントが実行されると，「END OF JOB」というメッセージが表示される。

DO

DOステートメントは，DOグループの冒頭に置かれるものであり，DOグループ内のステートメントを繰り返して実行するよう指定することもできる。

一般形式：

DOステートメントの形式には３つのタイプがある。これを図J.３に示す。

構文法細則：

1. ３つのどのタイプの場合も，DOステートメントは，ENDステートメントと対で，１つのDOグループを区切るのに使用される。DOグループ内のステートメントを繰り返し実行するよう指定しないのはタイプ１だけである。

2. タイプ３においては，変数または擬似変数は１つの要素を表わすものでなければならない。「変数」は，添字がついていても，修飾されていてもよいし，添字がついておりしかも修飾されているものでもよい。（擬似変数COMPLETION, COMPLEX, PRIORITY, およびSTRING を使用することはできない。）通常は，実数算術変数が使用されているが，以下の一般規則に従って行なわれた展開が有効なPL/Iプログラムになっている限り，どのようなタイプの変数でも使用することができる。「変数」がプログラム制御変数である場合には，「指定」の中でBYオプションならびにTOオプションを使用することができないという点に注意すること。

3. 「指定」の中の各々の式は，要素式でなければならない。

4. 「指定」に「BY式３」を記入せず「TO式２」を記入した場合には，「式３」は１であるとみなされる。

5. 「指定」に「TO式２」を記入しなかった場合には，WHILEオプションまたは他のステートメントによってDOグループの外に制御が渡されるまで，DOグループの実行が繰り返される。

6. 「指定」の中に「TO式２」も「BY式３」も記入しなかった場合には，DOグループが１回だけ実行され，制御変

```
タイプ1.  DO;
タイプ2.  DO WHILE (要素式);
タイプ3.  DO {擬似変数|変数} = 指定 〔,指定〕・・・;
  ここで,「指定」は次のような形をとる:
  式1 [TO 式2 〔BY 式3〕|BY 式3 〔TO 式2〕] 〔WHILE (式4)〕
```

図J.3 DOステートメントの一般形式

数は「式1」の値をとる。「WHILE 式4」を記入した場合には,「式4」が真でない限りこの実行は1回も行なわれない。

一般規則:

1. タイプ1においては,DOステートメントはDOグループの始まりを表わすだけであり,実行の繰返しは指定しない。

2. タイプ2においては,DOステートメントはDOステートメントの始まりを表わすと共に,次のように定義される実行の繰返しを指定する。

```
LABEL: DO WHILE (式);
       ステートメント-1
       .
       .
       .
       ステートメント-n
       END;
NEXT:  ステートメント /*STATEMENT
        FOLLOWING THE DO GROUP*/
```

上のコーディングは,次の展開と全く同じ内容を表わしている。

```
LABEL: IF  (式) THEN;
       ELSE GO TO NEXT;
       ステートメント-1
       .
       .
       .
       ステートメント-n
       GO TO LABEL;
NEXT:  ステートメント /*STATEMENT
        FOLLOWING THE DO GROUP*/
```

3. タイプ3においては,DOステートメントはDOグループの始まりを表わすと共に,次のように定義される制御された実行の繰返しを指定する。

```
LABEL: DO 変数 =
          式1
          TO 式2
            BY 式3
          WHILE (式4);
          ステートメント-1
          .
          .
          .
          ステートメント-m
LABEL1:END;
NEXT:  ステートメント
```

擬似変数でない変数に関する限り,上のコーディングは次の展開と全く同じ内容を表わしている。

```
LABEL:   p=ADDR (変数);
         e1= 式1;
         e2= 式2;
         e3= 式3;
         v=e1;
LABEL2:  IF (e3>=0)ε(v>e2)|
            (e3<0)ε(v<e2)
            THEN GO TO NEXT;
         IF (式4) THEN;
           ELSE GO TO NEXT;
         ステートメント-1
         .
         .
         .
         ステートメント-m
LABEL1:  v=v+e3;
         GO TO LABEL2;
NEXT:    ステートメント
```

ここで,pはコンパイラーの作成したポインターであり,vはコンパイラーの作成した(pを基底とする)基底付変数(「変数」と同じ属性をもつ)である。「e1」「e2」「e3」はコンパイラーの作成した変数(それぞれ,「式1」「式2」,「式3」の属性をもつ)である。初期値式(式1)の値が計算される直前に,制御変数の世代が,ループの外

で一度作成されている点に注意されたい。

上の展開に関する他の規則を以下に示す。

a. 上の展開は，１つの「指定」の結果を示したにすぎない。DOステートメントに２つ以上の「指定」がある場合には、NEXT というラベルのついたステートメントが次の「指定」に対応する展開の最初のステートメントになる。２番目の展開は，どの点から見ても，最初の展開に似ている。したがって，DOステートメントの中に２番目の「指定」がある場合には，２番目の展開は次のような形になるであろう。

```
NEXT：      e5= 式5 ;
            .
            .
            .
            v=e5;
LABEL3:     IF ・・・ THEN GO TO NEXT1;
            IF (式8) THEN;
              ELSE GO TO NEXT1;
            ステートメント－1
            .
            .
            .
            ステートメント－m
LABEL4:     v=v+e7;
            GO TO LABEL3;
NEXT1:      ステートメント
```

ステートメント１～mがプログラムの中で実際に２回繰り返して作成されるわけではないという点に注意すること。

b. WHILE オプションが指定されていない場合は，展開の中のステートメント－１の直前にあるＩＦステートメントは生成されない。

c. 「ＴＯ式２」が指定されていない場合は，ステートメント「ｅ２＝式２」ならびにLABEL２ によって識別されるＩＦステートメントは生成されない。

d. 「ＴＯ式２」および「ＢＹ式３」が指定されていない場合は，ｅ２およびｅ３に関係するすべてのステートメントならびにGO TO LABEL2 ステートメントは生成されない。

4. タイプ２およびタイプ３におけるWHILE オプションは，ステートメントの実行が１回済んだ後新たに最初からステートメントの実行を繰り返す前に，ここで指定されている要素式の値を計算し，必要に応じて，この要素式をビット・ストリングに変換するよう指定するためのものである。こうして得られたストリングの中に１の値をもつビットがあれば，DOグループのステートメントが実行される。すべてのビットが０である場合には，タイプ２においてはDOグループの実行が停止され，タイプ３においてはWHILE オプションをもつ「指定」に対応する実行だけが停止される。タイプ３では，この後，次の「指定」があればこの指定に対応する実行の繰返しが始まる。

5. 「指定」の中の「式１」は，制御変数（すなわち，「変数」または「擬似変数」）の初期値を表わすためのものである。「式３」は，DOグループに属するステートメントの実行が１回完了するたびに制御変数に加えるべき増分を表わすためのものである。「式２」は制御変数の終了値を表わすためのものである。ループが終わるたびに制御変数の値がテストされ，この値が「式１」と「式２」で示されている範囲の外にあると，直ちに，当該「指定」に対応するDOグループのステートメントの実行が停止される。最後の「指定」に対応する実行が終了すると，通常，DOグループの直ぐ後にあるステートメントに制御が渡される。

6. タイプ１のDOステートメントによってDOグループが構成されている場合にだけ（すなわち，実行の繰返しが指定されていない場合だけ），DOグループの外部からDOグループの内部に制御を移すことができる。したがって，繰返し実行されるDOグループにENTRY ステートメントがあってはならない。

7. ポインターで修飾されている制御変数または被制御・制御変数の世代は，初期値式（式１）の値が計算される直前に，ループの外で生成される。ループの中で制御変数のポインターを変更したり，制御変数を割り振ったりしてその制御変数の世代を変更しても，先の世代から得た制御変数のままでループの実行が続けられる。しかし，ループの中でのこの制御変数を参照すると，後の世代を参照することになる。この世代を解放すると誤りになる。

8. ネスティングの最大の深さは４９である。

END

END ステートメントは，ブロックおよびグループを終了させるためのものである。

一般形式：

END〔識別名〕;

構文法細則：

「識別名」は，ラベル定数または入口定数である。これは添字付の

名前であってはならない

一般規則：

1. ENDの後に識別名が記入されている場合には，そのラベル定数ないし入口定数をもつDO，BEGIN，またはPROCEDURE ステートメントのうち，ENDステートメントより前の最も近い位置にあるもので始まっている，未終了のグループないしブロックを終了させる。また，物理的にこのグループまたはブロックの中にある未だ終了していないグループまたはブロックも，このENDステートメントによって，終了する。

2. ENDの後に識別名が記入されていない場合には，対応するENDステートメントをもたないDO，BEGIN，またはPROCEDURE ステートメントのうちこのENDステートメントより前にあり，しかもこのENDステートメントに一番近いところにあるもので始まるグループないしブロックが，このENDステートメントによって，終了する。

3. あるプロシージャーのENDステートメントに制御が渡ると，このステートメントはRETURN ステートメントとして取り扱われる。

ENTRY

ENTRYステートメントは，プロシージャーの副次入口点を指定するためのものである。

一般形式：

```
入口定数： 〔入口定数：〕・・・
  ENTRY 〔（パラメーター〔，パラメーター〕・・・）〕
        〔RETURNS （属性リスト）〕
        〔IRREDUCIBLE|REDUCIBLE〕
        〔OPTIONS （オプション・リスト）〕；
```

構文法細則：

1. ENTRYステートメントにおいて，RETURNS オプションの「属性のリスト」に指定することのできる属性は算術，ストリング，ALIGNED，UNALIGNED，POINTER，OFFSET，AREA，FILE，EVENT，およびLABELだけである。ストリングには，VARYING 属性を与えることができる。エリアの名前が添字付の名前，修飾された名前，あるいは被定義の名前でない限り，OFFSET 属性でエリアを指定することができる。また，LABEL属性にラベル定数のリストを含めることもできる。エリアの大きさやストリングの長さを与える場合には，10進整数定数でこれらを指定しなければならない。

2. OPTIONS オプションの「オプションのリスト」は，コンパイラーごとに定義されている1つまたはそれ以上の追加オプションを指定するためのものである。これらの追加オプションを以下に示す。

```
{COBOL | FORTRAN}
〔NOMAP      〔（引数のリスト）〕〕
〔NOMAPIN    〔（引数のリスト）〕〕
〔NOMAPOUT   〔（引数のリスト）〕〕
```

オプションとオプションの間にはブランクを置く。オプションを記入する順序は任意である。

「引数のリスト」は，オプションを適用すべきパラメーターの名前のリストである。1つの引数のリストで指定できるパラメーターは64個までである。リストの中にパラメーターを記入する順序は任意である。パラメーターとパラメーターの間にはコンマまたはブランクを置く。引数のリストが指定されていない場合には，当該入口名と結び付いているすべてのパラメーターにこのオプションを適用するものとみなされる。

同一のOPTIONS の指定の中でNOMAP，NOMAPIN，およびNOMAPOUT を3つとも指定することができる。ただし，この指定は，指定または想定されている2つ以上の引数リストの中で同じパラメーターを含んでいてはならない。これらのオプションの用途については，「第17章： 言語間の連絡機能」を参照のこと。

一般規則：

1. 副次入口点のパラメーターとこの入口点に渡される引数との間に設定される関係は，主入口点のパラメーターと引数との間に設定される関係と全く同じである。この点について詳しくは，「第9章： サブルーチンと関数」を参照のこと。

2. 構文法細則1で説明したように，ENTRYステートメントで指定されている入口点を用いてプロシージャーを関数として呼び出した後そのプロシージャーから返される値の特性は，ENTRYステートメントで指定されている属性によって決まる。プロシージャーから返される値（すなわち，RETURN ステートメントの中の式の値）は，必要に応じて，指定されている属性に合うよう変換される。入口点において属性が指定されていない場合には，その入口点を呼び出すのに用いられた入口名の頭文字にもとづいて省略時属性が適用される。

3. 1つのENTRYステートメントに2つ以上の名前がある場合には，各々の名前は，同じパラメーター・リストと同じ属性の明示的指定とをもつ別々のENTRYステートメントに対する1個の名前であるとみなされる。たとえば，次のようなステートメントがあるものとする。

```
A： I： ENTRY；
```

このステートメントは，事実上，以下のステートメントと同じである。

```
A： ENTRY；
```

I：　ENTRY；

返される値の属性が明示的に指定されていないので，プロシージャーから返される値の特性は，AとIのどちらによって入口点が呼び出されたかによって決まる。

4. ENTRYステートメントは，このステートメントで定義されている副次入口点をもつプロシージャーの内部になければならないが，このプロシージャーに含まれているブロックの内部にあってはならない。また，実行の繰返しを指定しているDOグループの中にENTRYステートメントがあってはならない。

5. 通常の逐次的な制御の流れがENTRYステートメントに出会うと，制御はENTRYステートメントを迂回する。

6. IRREDUCIBLEとREDUCIBLEは最適化に関するオプションである。DOS PL/I 最適化コンパイラーによって処理されるプログラムの中でこれらのオプションが使用されている場合，これらのオプションに構文法上のエラーがないかどうかチェックされるだけで，これらのオプションは無視される。この種のプログラムにこれらのオプションがあっても誤りではない。

7. 上述のオプションの意味を以下に示す。

COBOL：

このPL/I プロシージャーは，この入口点では，COBOLサブプログラムによってだけ，呼び出される。

FORTRAN：

このPL/I プロシージャーは，この入口点では，FORTRAN ルーチンによってだけ，呼び出される。

NOMAP，NOMAPIN，NOMAPOUT：

これらのオプションは，COBOLないしFORTRANとPL/I とのインターフェースにおいて，データ集合体の自動的な取扱いが行なわれないようにするためのものである。各オプションの引数のリストで，各々のオプションを適用すべきパラメーターを指定することができる。あるオプションの引数のリストが指定されていない場合には，当該入口名と結び付いているすべてのパラメーターにそのオプションを適用するものとみなされる。

FORMAT

FORMATステートメントは，転送するデータの形式を制御するために編集指示転送ステートメントにおいて使用することのできる形式リストを指定するためのものである。

一般形式：

ラベル：〔ラベル：〕・・・FORMAT（形式リスト）；

構文法細則：

1. 「形式リスト」は，編集指示転送のための形式リストの指示に関する規則に従って指定しなければならない。この規則については，「第１０章：　入力と出力」を参照のこと。

2. FORMATステートメントでは，「ラベル」を少なくとも１つ指定しなければならない。いくつかあるラベルの内の１つ（または，いくつかあるラベルの内の１つの値を表わすラベル変数ないし関数参照）は，遠隔形式項目の中にあるステートメント・ラベル標識である。

一般規則：

1. GETまたはPUTステートメントの編集指示データ指定の形式リストの中に遠隔形式項目Rを入れることができる。形式リストのRで表わされる部分は，Rで指定されているステートメント・ラベルで識別されるFORMATステートメントによって，補なわれなければならない。

2. １で述べた遠隔形式項目とFORMATステートメントは，同じブロックの内部になければならない。

3. FORMATステートメントに条件接頭語をつける場合には，そのFORMATステートメントを参照しているGETないしPUTステートメントの条件接頭語と同じものをつけなければならない。

4. 通常の逐次的な制御の流れがFORMATステートメントに出会うと，制御はこのステートメントを迂回する。また，このステートメントのステートメント・ラベルに関してCHECK条件が引き起されることはない。

5. GO TOステートメントを用いてFORMATステートメントに制御を渡すのは，誤りである。

FREE

FREEステートメントは，ここで指定されている基底付変数ないし被制御変数に割り振られている記憶域を解放するためのものである。ここで指定されているのが被制御変数である場合には，次に新しい割振りが使用可能になり，これ以後のこの識別名への参照はその割振りへの参照となる。

一般形式：

FREE オプション〔，オプション〕・・・；

ここで，「オプション」は次のいずれかの形をとる。

オプション1：
　識別名
オプション2：
　〔ロケーター修飾子－＞〕・・・
　基底付変数識別名
　〔IN（要素エリア変数）〕

構文法細則：

1. オプション1の「識別名」は，被制御記憶域クラスの要素，配列，ないし大構造である。

2. オプション2の「基底付変数識別名」は，添字のついていないレベル1の基底付変数でなければならない。

3. 1つのステートメントの中でオプション1とオプション2を使用してもよい。

一般規則：

1. FREEステートメントが実行される時点で，指定されている基底付でない識別名に記憶域が全く割り振られていない場合には，何も行なわれない。

一般規則2～5は，オプション2にだけ適用される。

2. 基底付変数が明示的に修飾されていない場合には，この基底付変数といっしょに明示的に宣言されているロケーターを用いて，記憶域の解放すべき部分にあるデータの世代が識別される。ロケーターが宣言されていない場合には，このステートメントは誤りになる。

3. 解放される記憶域の大きさは，基底付変数の属性（指定があれば，記憶域が解放される時点の限界および／または長さも含めて）によって決まる。こうして決定される大きさと割り振られている大きさとが一致するよう図るのはユーザーの側の責任である。基底付変数の記憶域が割り振られていない場合には，結果はわからない。

4. 基底付変数を使って記憶域を解放する場合，その記憶域は同じデータ属性をもつ基底付変数に割り振られているものでなければならない。

5. 解放すべき記憶域がエリアの中に割り振られている場合には，INオプションを指定するか，これを暗示する。基底付変数がエリアの中に割り振られていない場合には，INオプションを指定することはできない。エリア割当てによって目標エリアの中に基底付記憶域が割り振られるという点に注意すること。この目標エリアを指定しているINオプションによって，このような割振りを解放することができる。

GET

GETステートメントは，次のいずれかの方法で使用されるSTREAM転送ステートメントである。

1. 外部記憶媒体（データ・セット）にあるデータを計算機内部の1つ以上の受入れ側フィールド（変数）に割り当てる。

2. 計算機内部の記憶域（文字ストリング変数）にあるデータを計算機内部の1つ以上の受入れ側フィールド（変数）に割り当てる。

一般形式：

GET オプション・リスト；

「オプション・リスト」の形式は，次の通りである。

〔FILE（ファイル式）|
STRING（文字ストリング名）〕
〔データ指定〕〔COPY〔（ファイル式）〕
　〔SKIP〔（式）〕〕

一般規則：

1. FILEオプションもSTRINGオプションも指定されていない場合には，FILE(SYSIN)であるものと想定される。

2. SKIPオプションを指定しない場合には，「データ指定」を少なくとも1つ記入しなければならない。

3. オプションを記入する順序は任意である。

4. FILEオプションの「ファイル式」では，割り当てられる値を収容しているデータ・セットと結び付けられた（開放のときに）ファイルを指定する。このファイルはSTREAM INPUTファイルでなければならない。

   COPYオプションの「ファイル式」では，値を受け取ることになっているデータ・セットと結び付けられているファイルを指定する。このファイルは，STREAM OUTPUT ファイルでなければならない。

5. 「文字ストリング名」では，データ・リストに割り当てられるデータを収容している文字ストリングを指定する。この名前は組込み関数への参照であってもよい。このオプションのもとでのGET操作は，各々，指定されているストリングの最初から始められる。このストリングの文字数がデータ指定で指定されている文字総数よりも小さい場合には，ERROR条件が引き起される。

6. データ指示転送のもとでSTRINGオプションを使用する場合には，ストリングの中の識別名と同じ識別名がデータ指定の中になければ，ERROR条件が引き起される。

7. 「データ指定」については，「第11章：ストリーム方式の転送」を参照のこと。

8. FILEオプションで指定されているファイルが開かれていない場合には，ストリーム式入力転送をするために，そのファイルが暗示的に開かれる。

9. COPYオプションをSTRING オプションといっしょに使用することはできない。COPYオプションは，原データの流れを，読み取ったままの形で何の変更も加えることなく，指定されているファイルに書き込むよう指定するためのものである。入力の流れに新しいレコードが出て来るたびに，COPY ファイルのレコードも新しくなる。省略時のファイルは，標準ファイルのSYSPRINT である。

COPYオプションで指定されているファイルが開かれていない場合には，ストリーム式出力転送をするためにこのファイルが暗示的に開かれる。

10. COPYオプションをもつGETステートメントを実行している間に割込みが発生し，このためにあるオン・ユニットに入ったが，このオン・ユニットにおいて同じファイルを対象とする別のGETステートメントが実行されており，さらにこの後オン・ユニットから割込みの発生したステートメントに制御が戻された，という場合には，COPYオプションが指定されていないものとしてこのGETステートメントの実行が続行される。オン・ユニットにおいて，COPY オプションで指定されているファイルを対象とするPUTステートメントが実行された場合には，転送されるデータの位置が最後に転送されたCOPY データ項目の直ぐ後になるとは限らない。

11. SKIPオプションをSTRING オプションといっしょに指定することはできない。SKIPオプションは，当該ファイルにおける新しい「現時点の行」を定義するためのものである。SKIPオプションで式が指定されている場合には，この式が整数（w）に変換される。wは0より大きくなければならない。wが0より大きくない場合は，コンパイラーがwの値を1にする。データ・セットは，「現時点の行」から数えてw番目の行に位置設定される。式が指定されていない場合には，SKIP(1) であるとみなされる。SKIPオプションは，つねに，データが転送される前に実行される。

GO TO

省略形：　GOTO

GO TOステートメントは，ここで指定されているラベルをもつステートメントに制御を渡すためのものである。

一般形式：

{GOTO}　要素ラベル式；

一般規則：

1. 「要素ラベル式」はラベル定数，ラベル変数，またはラベル値を返す関数参照である。GO TOステートメントを実行するたびにラベル式の値が変っていることもあるので，いつも同じステートメントに制御が渡されるとは限らない。

2. GO TOステートメントを用いて非活用状態にあるブロックに制御を渡すことはできない。

3. DOグループの中で実行の繰返しが指定されている場合には，GO TOステートメントを用いてDOグループの外部から，DOグループの内部にあるステートメントに制御を渡すことはできない。ただし，DOグループの内部で呼び出されたプロシージャーないしオン・ユニットをGO TOステートメントによって終了させることはできる。

4. あるブロックの中にあるGO TOステートメントによってそのブロックに含まれていない場所に制御が渡されると，そのブロックは終了する。また，制御が渡った場所が，上述の終了させられたブロックを直接活用化したのではないブロックに含まれている場合は，活用状態化の順序から見て両ブロックの間にあるすべてのブロックも終了する（この例と詳細については，「第8章：　記憶域の制御」を参照のこと）。あるGO TOステートメントによって1つ以上のブロックが終了すると，これらのブロックが通常の形で終了したのと同様にして，各種条件が復元され，自動変数が解放される。

5. 関数として呼び出されたプロシージャーの中にあるGO TOステートメントによってその外に制御が渡された場合には，対応する関数参照を含む式の値の計算が中止される。

IF

IFステートメントは，指定されている式の値をテストし，このテストの結果にもとづいて実行の流れを規制するためのものである。

一般形式：

IF 要素式
　　THEN ユニット－1
　　［ELSE ユニット－2］

構文法細則：

1. 各々のユニットは，1つのステートメント（DO,END, PROCEDURE,BEGIN,DECLARE,DEFAULT, FORMAT,ENTRY を除く），DOグループ，または開始ブロックのいずれかである。

2. IFステートメントそのものの終りにはセミコロンを記入しない。しかし，指定する各々の「ユニット」の終りにはセミコロンを記入しなければならない。

3. 各々の「ユニット」には，ラベルや条件接頭語がついていてもよい。

一般規則：

1. 要素式の値が計算され，必要に応じて，要素式がビット・ストリングに変換される。ELSE 文節（ELSE とその後の「ユニット」）が指定されている場合には，この後次のような措置がとられる。

   ストリングの中に値1をもつビットがある場合には，「ユニット－1」が実行され，次いで，IFステートメントの次のステートメントに制御が渡される。ストリングの中のすべてのビットの値が0である場合には，「ユニット－1」は実行されずに「ユニット－2」が実行され，この後，次のステートメントに制御が渡される。

   ELSE 文節が指定されていない場合には，次のような措置がとられる。

   ストリングの中に値1をもつビットがある場合には，「ユニット－1」が実行され，次いで，IFステートメントの次のステートメントに制御が渡される。すべてのビットが0である場合には，「ユニット－1」は実行されずに次のステートメントに制御が渡される。

   各々の「ユニット」の中に制御の移動を行なうためのステートメント（たとえば，GO TO）があってもよい。ということは，IFステートメントの通常の制御の流れを変更してもよいということである。

   要素式に配列変数または構造変数を記入することができるのは，その変数が，要素値を返す関数へ渡される引数である場合だけである。

2. IFステートメントをネストすることができる。つまり，一方の「ユニット」または双方の「ユニット」にIFステートメントが含まれていてもよい。ELSE 文節は，つねに，同じブロックないしDOグループの中にあってELSE と対になっていない一番内側のIFと結びつく。したがって，所要の制御の流れを指定するために，ナル・ステートメントをもつELSE を使用する必要があることもある。ネスティングの深さは，最高49まで許される。

LOCATE

LOCATEステートメントは，BUFFERED OUTPUT ファイルに適用されるもので，緩衝域の中で基底付変数の割振りを行なうのに使用される。また，このステートメントを用いて，すでに緩衝域の中に割り振られている基底付変数を転送することもできる。

一般形式：

```
LOCATE 変数 FILE（ファイル式）
〔SET（ポインター変数）〕
〔KEYFROM（式）〕；
```

構文法細則：

1. オプションを記入する順序は任意である。

2. 「変数」は，添字のついていないレベル1の基底付変数でなければならない。

一般規則：

1. FILE オプションは，対象となるファイルを指定するためのものである。このオプションは必ず指定しなければならない。

2. LOCATEステートメントが実行されると，指定されている基底付変数が緩衝域の中に割り振られる。また，基底付変数の（REFERオプションで指定されている）構成要素が初期設定される。さらに，ポインター値が，SET オプションで指定されているポインター変数に割り当てられる。SET オプションが指定されていない場合には，ポインター値が，基底付変数の宣言で指定されているポインター変数に割り当てられる。このポインター値は緩衝域の中のレコードを指している。LOCATE ステートメントを実行した後，当該ファイルに対する次のLOCATE 操作，WRITE操作，またはCLOSE操作の直前に行なわれるデータ・セットへの転送に備えて，基底付変数に値を割り当てることができる。

3. KEYFROM オプションが指定されている場合には，式の値は文字ストリングに変換され，後にレコードが書き込まれるときにレコードのキーとして使用される。

4. FILE オプションで指定されているファイルが開かれていない場合には，このファイルは自動的に開かれる。したがって，このファイルを参照しているOPEN ステートメントがLOCATEステートメントの前にあるのと同じことになる。ただし，このファイルの宣言においてRECORD 属性とOUTPUT 属性とを指定しておかなければならない。さもないと，UNDEFINEDFILE 条件が引き起される。

5. 当該データ・セットの編成がCONSECUTIVE である場合に限り，LOCATE ないしLOCATE SETステートメントを用いて行なわれるSEQUENTIAL BUFFEREDファイルのレコードの転送は，自動的に，プログラム内の他

のステートメントの実行と並行して進められる。

ナル・ステートメント

ナル・ステートメントは，何も行なわず，またステートメントの実行順序も変更しないステートメントである。ナル・ステートメントのラベルに関してCHECK条件を解禁してある場合には，ナル・ステートメントに制御が渡るたびにCHECK条件が引き起される。

一般形式：

〔ラベル：〕・・・；

あるナル・ステートメントのラベルとその直後のステートメントのラベルとを比較しても等しくはならない，という点に注意すること。たとえば，次のようなコーディングがあるものとする。

```
A：；
B：  X＝T；
```

この場合，ラベルAとラベルBの比較結果は等しくない。

ON

ONステートメントは，ここで指定されている例外条件に起因する割込みが発生したときに，どのような措置（プログラマーの定義になる措置またはシステムの標準措置）をとるかを指定するためのものである。

一般形式：

ON 条件〔SNAP〕{SYSTEM；｜ オン・ユニット}

構文法細則：

1. 「条件」は，「H節： 条件」に記載されているどの条件でもよい。

2. 「オン・ユニット」自体の終りにセミコロンを記入しなければならないから，一般形式の「オン・ユニット」の終りにはセミコロンを示してない。SYSTEMの後には，セミコロンを記入しなければならない。

一般規則：

1. 指定されている条件に起因する割込みの処理方法は，ONステートメントによって決まる。すなわち，システムの標準方式でこの割込みを処理するか，プログラマーの指定した方式でこの割込みを処理するかは，ONステートメント内の処置の指定に応じて，次のようにして決められる。

   a. ONステートメントの処置の指定がSYSTEMである場合には，システムの標準処置がとられる。システムの標準処置は，多くの場合，メッセージを印刷しERROR条件を引き起すことだけである。しかし，どの条件についてもシステムの標準処置が同じであるというわけではない。各々の条件に対するシステムの標準処置については，「H節： ON条件」を参照のこと。（ある割込みが発生したとき，この割込みの原因である条件のためのONステートメントが効力をもっていない場合には，つねに，システムの標準処置がとられることに注意していただきたい。）

   b. 処置の指定が「オン・ユニット」である場合には，プログラマーの側で割込み処理のための独自の処置（オン・ユニット自体の中のステートメントによって定義された処置）を与えてある筈である。ONステートメントが実行されるときに，そのオン・ユニットが実行されるわけではない。オン・ユニットは，指定されている条件に起因する割込みが発生したときに（あるいは，SIGNALステートメントによって信号が出された条件に起因する割込みが発生して初めて），実行されるのである。

2. あるブロックの中にあるONステートメントが実行され，その結果その処置の指定（「オン・ユニット」またはSYSTEM)が設定されると，そのブロック，およびそのブロックによって開始された活用化の系列にあるすべてのブロックにわたって，この処置の指定が効力をもちつづける。ただし，別のONステートメントまたはREVERTステートメントが実行されると，次のようにして先の処置の指定は効力を失なう。

   a. 後のONステートメントで先のONステートメントと同じ条件が指定されており，しかも，先のONステートメントを含むブロックによって開始された活用化の切替系列にあるブロックの中で後のONステートメントが実行された場合には，先のONステートメントの処置の指定は一時停止される（積み重ねられる）。この後，REVERT ステートメントが実行されるか，後のONステートメントを含むブロックが終了すると，先のONステートメントの処置の指定を復元することができる。

   b. 後のONステートメントと先のONステートメントが共に同一ブロックの同一の呼出しの内部にある場合には，先のONステートメントは完全に無効になる。

3. コンパイラーは，つねに，オン・ユニットを，そのオン・ユニットが入っているブロックの内部にあるプロシージャーとして取り扱う。（概念上は，オン・ユニットはPROCEDUREステートメントとENDステートメントに囲まれている。）

あるオン・ユニットの中で参照される名前はすべて，そのオン・ユニットを指定しているONステートメントが実行された環境において知られている名前であり，割込みが発生した環境において知られている名前ではない。

4. ある条件が引き起された時点においてその条件が解禁状態にある場合に限り，実行中に引き起された条件は，割込みを発生させる。

   a. 以下の条件は省略時解釈によって解禁状態にされる。
   AREA, OVERFLOW, FIXEDOVERFLOW, UNDERFLOW, ZERODIVIDE, CONVERSION, すべての入出力条件，CONDITION, FINISH, ERROR。

   b. 以下の条件は，省略時解釈によって禁止状態にされる。
   SIZE, STRINGSIZE, STRINGRANGE, SUBSCRIPTRANGE, CHECK。

   c. 条件接頭語を用いて，以下の条件を解禁状態に置くか，禁止状態に置くか指定することができる。OVERFLOW, FIXEDOVERFLOW, UNDERFLOW, ZERODIVIDE, CONVERSION, SIZE, STRINGSIZE, STRINGRANGE, SUBSCRIPTRANGE, CHECK。

5. オン・ユニットが1つのステートメントだけで構成されている場合には，オン・ユニットの中で遠隔形式指定を参照することができる。

6. SNAP が指定されている場合には，指定されている条件が引き起され，割込みが生じると，呼出し追跡がリストされる。すなわち，割込み発生の時点において活用状態にあるすべてのブロックがSYSPRINTにリストされる。

OPEN

OPEN ステートメントはファイル名とデータ・セットとを結び付けるためのものである。また，開くべきファイルの宣言で指定されている属性が不充分である場合には，OPEN ステートメントでそのファイルの属性を補充することができる。

一般形式：

```
OPEN FILE （ファイル式）〔オプションのグループ〕
〔,FILE （ファイル式）〔オプションのグループ〕〕…;
```

ここで 「オプションのグループ」は次のような形をとる。

```
〔INPUT｜OUTPUT〕
〔LINESIZE（要素式）〕
〔PAGESIZE（要素式）〕
〔TITLE （要素式）〕
```

構文法細則：

1. INPUTオプションあるいはOUTPUTオプションでは，ファイルの宣言で指定されている属性を補充するための属性を指定する。

2. あるファイルの「オプションのグループ」の中のオプションおよびFILE オプションを記入する順序は任意である。

3. 「ファイル式」では，データ・セットと結び付けるべきファイルの名前を指定する。1つのOPEN ステートメントでいくつかのファイルを開くことができる。

一般規則：

1. すでに開かれているファイルを開くよう指定しても，何も行なわれない。

2. CONSECUTIVE 編成のUNBUFFEREDファイルに対してだけ，INPUTオプションあるいはOUTPUT オプションを指定することができる。

3. LINESIZE オプションは，STREAM OUTPUT ファイルに対してだけ指定することができる。この式を計算して整数に変換したものが，当該ファイルに対する次の操作の間，行の長さとして用いられる。印刷形式項目および制御形式項目を使うか，GET ないしPUTステートメントのオプションを使えば，新しい行を開始させることができる。新しい行を開始するための明示的に指定された措置がとられる前に，ある行の終りより右側の場所にファイルの位置を設定しようとすると，自動的に新しい行が開始され，この新しい行の最初にファイルの位置が設定される。当コンパイラーでは，以下の数値が適用される。

```
行の最大の長さ：
    F形式またはU形式：  32,759
    V形式          ：  32,751

行の最小の長さ：
    FまたはU                 1
    V： PRINTファイル        9
        PRINT以外のファイル 10

省略時の行の長さ           120
```

INPUTファイルに対してLINESIZE オプションを指定することはできない。GET ステートメントのSKIPオプションが出て来るたびにとられる行の長さは，当該データ・セットを作成するとき使用された行の長さである。データ・セットの作成のとき行の長さが使用されていない場合には，論理レコードの「現時点の長さ」（V形式レコードに関しては，これから制御バイトを引いた値）が行の長さであるとみなされる。

4. PAGESIZE オプションは，STREAM 属性とPRINT属性をもつファイルに対してだけ指定することができる。このオプションで指定されている要素式の値が計算され，その値が

整数に変換される。この整数は1ページあたりの最大の行数を表わす。この後行なわれるPRINTファイルへの転送の間に，PAGE 形式項目を使うかまたはPUTステートメントのPAGE オプションによって，新しいページを開始することができる。新しいページを開始するための措置がとられる前に，あるページが一杯になりしかも印刷すべきデータがまだ残っている場合には，ENDPAGE 条件が引き起される。当コンパイラーにおいては，以下の数値が適用される。

ページの最大の大きさ	32,767
ページの最小の大きさ	1
ページの省略時の大きさ	60

5. STREAMファイルが開かれると，丁度ゼロ番目のレコードの走査が完了した位置に，概念上このファイルが設定される。つまり，このファイルは，最初のGETないしPUTステートメントによってアクセスされるレコードの直前にある想像上のレコードの終りに位置設定される。したがって，最初のGETないしPUTステートメントのオプションまたは形式項目によって，n行スキップした後初めてレコードにアクセスするよう，指定されている場合には，ファイルはn番目のレコードの始めに位置設定される。PRINTファイルにおいては，「ゼロ番目のレコード」はページの大きさに等しい行番号をもつ概念上の行を指す。最初のPUT ステートメントに，PAGEオプションないし形式項目，あるいはゼロ行スキップするようにとの指示以外の指定があると，ENDPAGE 条件が引き起される。最初のPUT ステートメントの指定がPAGEオプションまたは形式項目だけであったり，ゼロ行スキップするようにとの指示だけであったりした場合には，ファイルには何の変化もない。

6. 次のようなファイルに対してTITLEオプションを指定することはできない。CONSECUTIVE編成のUNBUFFEREDファイル。磁気テープ装置と結び付けられているファイル。MEDIUMオプションで明示的に指定されたユニット・レコード装置と結び付けられているファイル。

TITLEオプションが指定されている場合には，「要素式」が，必要に応じて，文字ストリングに変換される。この文字ストリングの最初の7文字は，（DLBL ないし TLBLジョブ制御ステートメントのファイル名を介して，）当該ファイルに結び付けられるデータ・セットを表わす。このオプションが指定されていない場合には，ファイル名の最初の7文字（必要に応じて，パディングないし切捨てをする）がDLBLないしTLBL ファイル名であるとみなされる。これが外部名の場合と同じ切捨てでない点に注意すること。ファイルがパラメーターである場合には，パラメーター識別名によって表わされるファイル名ではなく，引数によって表わされるファイル名が使用される。

PROCEDURE

省略形：　PROC

PROCEDUREステートメントの機能は次に示す通りである。

- プロシージャーの冒頭に置かれる。
- このプロシージャーへの主入口点を定義する。
- 主入口点で指定すべきパラメーターがある場合には，このパラメーターを指定する。
- プロシージャーがもつある種の特別の特性を指定する。
- プロシージャーが関数として主入口点を用いて呼び出された場合に，そのプロシージャーが返す値の属性を指定する。

一般形式：

```
入口定数： 〔入口定数〕・・・
  PROCEDURE〔（パラメーター〔，パラメーター〕・・・）〕
  〔OPTIONS（オプション・リスト）〕
  〔RECURSIVE〕〔RETURNS（データ属性）〕
  〔ORDER|REORDER〕〔IRREDUCIBLE|
   REDUCIBLE〕;
```

構文法細則：

1. RETURNS オプションの「データ属性」では，このプロシージャーが関数として主入口点を用いて呼び出されたときこのプロシージャーが返す値の属性を指定する。ここで指定することのできる属性は，算術，ストリング，ALIGNED，UNALIGNED, POINTER, OFFSET, AREA, FILE, EVENT，およびLABELだけである。ストリングにはVARYING 属性を与えることができる。OFFSET属性ではエリア名を指定することができる。ただし，この名前は，修飾された名前，添字付の名前，あるいは被定義名であってはならない。また，LABEL属性ではラベル定数のリストを指定することができる。エリアの大きさあるいはストリングの長さは，10進整数定数で指定しなければならない。

2. OPTIONS, RECURSIVE, RETURNS, ORDER または REORDER，および，IRREDUCIBLE または REDUCIBLE を記入する順序は任意であるが，それぞれの間にはブランクを置く。

3. OPTIONS オプションの「オプション・リスト」は，コンパイラーごとに定義される追加オプションを1つ以上指定するためのものである。当コンパイラーの追加オプションは，次の通りである。

```
{MAIN|COBOL|FORTRAN}
〔NOMAP 〔（引数のリスト）〕〕
〔NOMAPIN 〔（引数のリスト）〕〕
〔NOMAPOUT 〔（引数のリスト）〕〕
```

〔REENTRANT〕

オプションとオプションの間にはブランクまたはコンマを置く。オプションを記入する順序は任意である。

「引数のリスト」は，当該オプションが適用されるパラメーターの名前のリストである。1つの「引数のリスト」の中で指定できるパラメーターは，64までである。パラメーターの名前を記入する順序は任意である。名前と名前の間にはコンマまたはブランクを置く。「引数のリスト」が指定されていない場合には，当該入口名と結び付いているすべてのパラメーターにこのオプションを適用するものとみなされる。

同一のOPTIONS 属性指定の中にNOMAP，NOMAPIN，およびNOMAPOUT を3つとも記入することができる。ただし，指定あるいは想定されている2つ以上の引数のリストの中に同じパラメーターがあってはならない。

COBOL，FORTRAN，NOMAP，NOMAPIN，NOMAPOUTの各オプションの使い方については，「第17章： 言語間の連絡機能」を参照のこと。

注： REENTRANTオプションは，多重タスク操作を行なうことのできるPL/Iコンパイラーの場合に使用される。DOS PL/I 最適化コンパイラーではこの機能が与えられていないので，REENTRANTオプションが指定されていても何の効果ももたない。この指定は，構文法にかなっているか否かテストされるだけで，無視されてしまう。

一般規則：

1. 当該プロシージャーが呼び出されると，当該プロシージャーに渡される引数と，呼び出されたプロシージャーの中でこの引数を表わすパラメーターとの間に関係が設定される。この点については，「第9章： サブルーチンと関数」を参照のこと。

2. OPTIONS は，外部プロシージャーに対してだけ指定することができる。少なくとも1つの外部プロシージャーにはOPTIONS(MAIN) と言う指定がなければならない。2つ以上の外部プロシージャーに対してこの指定が行なわれている場合には，オペレーティング・システムは，物理的に最初に現われたプロシージャーを呼び出す。

   以下では，上述の他のオプションの意味について説明する。

   COBOL：

   当該PL/Iプロシージャーは，COBOLサブプログラムによってだけ，主入口点を使って呼び出される。

   FORTRAN：

   当該PL/Iプロシージャーは，FORTRAN ルーチンによってだけ，主入口点を使って呼び出される。

   NOMAP，NOMAPIN，NOMAPOUT ：

   これらのオプションは，COBOLないしFORTRANとPL/Iとの間のインターフェースにおいて，データ集合体が自動的に取り扱われるのを防止するためのものである。各々のオプションの「引数のリスト」では，各オプションが適用されるパラメーターを指定することができる。あるオプションに対して「引数のリスト」が指定されていない場合には，当該入口名と結び付いているすべてのパラメーターにこのオプションを適用すべきものとみなされる。

   REENTRANT：

   多重タスク操作が可能なコンパイラーのもとでプロシージャーが実行される場合，REENTRANTオプションが指定されていると，コンパイルの済んだプロシージャーには再入可能という性質が与えられる。DOS PL/I最適化コンパイラーのもとでのコンパイレーションにおいては，このオプションは何の効果ももたない。

3. プロシージャーが回帰的に呼び出される場合には，RECURSIVE と指定しなければならない。プロシージャーが回帰的に呼び出されるということは，プロシージャーが未だ活用状態にある間にこれを再び活用化することである。RECURSIVE の指定は，当該プロシージャーがもつすべての入口点（主入口点ならびに副次入口点）に適用される。RECURSIVEは，これが宣言されているプロシージャーに対してだけ適用される。

4. RETURNS オプションの「データ属性」は，当該プロシージャーが関数として主入口点を用いて呼び出された場合に当該プロシージャーが返す値の属性を指定するためのものである。呼び出されるプロシージャーのRETURN ステートメントで指定されている値は，呼出しプロシージャーに返される前に，ここで指定されている属性に合致するよう変換される。

   RETURNS オプションが指定されていない場合には，省略時属性が適用される。この場合，入口点の名前（当該プロシージャーを呼び出すのに使用された入口定数）にもとづいて省略時の基数，精度，およびスケールが決められる。（入口点はいくつかの入口定数をもつことができるので，入口定数に応じて，省略時の基数，精度，およびスケールは異なる。）

5. ORDERとREORDERは，最適化に関するオプションである。すなわち，ORDERとREORDERは当該ブロックの最適化をどの程度まで進めるか指定するためのものである。一般に，ORDER と指定すると，あるブロッ

クを実行している間に入る計算関係のオン・ユニットにおいて，そのブロックでセットされたすべての変数の最新の値を使用することができる，という程度まで最適化が進められる。REORDER のもとでは，これ以上に最適化を進めることができる。すなわち，REORDER を指定した場合は，あるブロックでセットされた変数の値は，そのブロックを実行している間に入ったオン・ユニットの中でも一番新しい時点で割り当てられた値であるとは保証されない。いずれのオプションも指定されていないと，ORDER であるとみなされる。ただし，含まれているブロックは，その中で明示的にORDERが指定されていない限り，REORDERという指定を引き継ぐ。

6. IRREDUCIBLEとREDUCIBLEは，最適化に関するオプションである。DOS PL/I 最適化コンパイラーで処理されるプログラムの中でこのオプションが使用されていた場合，その指定に構文法上のエラーがないかどうかチェックされるだけで，その指定は無視される。この種のプログラムでこのようなオプションが指定されていても誤りではない。

7. 1つのPROCEDUREステートメントが2つ以上の入口定数をもっている場合は，最初の名前がそのステートメントの唯一のラベルであるとみなすことができる。2番目以降の入口定数は，各々，そのPROCEDUREステートメントで指定されているのと同じパラメーターのリストをもつ別々のENTRYステートメントであるとみなすことができる。たとえば，次のようなステートメントがあるものとしよう。

```
A:  I:  PROCEDURE (X);
```

このステートメントは，事実上，以下のステートメントと同じ内容を表わしている。

```
A:  PROCEDURE (X);
I:  ENTRY (X);
```

返される値の属性が明示的に指定されていないので，このプロシージャーが返す値の特性は，このプロシージャーがAとIのどちらを使って呼び出されたかによって異なる。

PUT

PUTステートメントは，以下のいずれかの方法で使用することのできるSTREAM転送ステートメントである。

1. 内部記憶域の中の1つ以上の場所にある値を外部媒体上のあるデータ・セットに転送することができる。

2. 内部記憶域の中の1つ以上の場所にある値を同じ内部記憶域の中の（文字ストリング変数で表わされる）受入れ側フィールドに割り当てることができる。

一般形式：

```
PUT 〔FILE（ファイル式）〕
    〔STRING（文字ストリング変数）〕
    〔データ指定〕
    ⎧PAGE〔LINE（要素式）〕⎫
    ⎨SKIP〔（要素式）〕    ⎬
    ⎩LINE（要素式）        ⎭ ;
```

構文法細則：

1. FILEオプションもSTRING オプションも指定されていない場合には，FILE(SYSPRINT)であるものとみなされる。この種のPUTステートメントがSYSPRINTという識別名の宣言の有効範囲にある場合には，SYSPRINTはFILE STREAM OUTPUTとして宣言されていなければならない。この種のPUTステートメントがSYSPRINTの宣言の有効範囲にない場合には，SYSPRINTは標準のシステム出力ファイルである。

2. FILEオプションは，外部記憶媒体上のデータ・セットへの転送について指定するためのものである。このオプションで指定されているファイル式は，値を受け取ることになっているデータ・セットと結び付けられている(暗示的な開放または明示的な開放によって）ファイルの名前である。このファイルは，OUTPUT属性とSTREAM属性をもつものでなければならない。

3. STRING オプションは，内部記憶域内の（「データ指定」の変数または式によって表わされる）場所から（「文字ストリング変数」によって表わされる）文字ストリングへの転送について指定するためのものである。「文字ストリング変数」は，ストリング擬似変数であってもよい（STRINGはのぞく）。

4. 「データ指定」については，「第11章： ストリーム方式の転送」を参照のこと。

5. STRINGオプションといっしょにPAGE, SKIP, LINEの各オプションを指定することはできない。

6. オプションを記入する順序は任意である。ただし，少なくとも1つのオプションを指定しなければならない。

一般規則：

1. FILEオプションが指定されている場合，「ファイル式」で参照しているファイルがまだ開かれていなくてもそのファイルがOUTPUT 属性をもつものとして明示的に宣言されているならば，そのファイルは暗示的に開かれる。これ以外の場合には，UNDEFINEDFILE 条件が引き起される。

2. STRINGオプションが指定されている場合は，適宜変換が行なわれた後，PUT操作における値の割当てがストリングの最初から（左端の文字から）始められる。ブランクや区切り文字は通常通り挿入される。ストリングが短

かすぎてデータを収容し切れない場合には，ERROR条件が引き起される。

3. PAGEオプションおよびLINEオプションは，PRINTファイルに対してだけ指定することができる。これらのオプションはいずれも，与えられている「データ指定」によって定義されている値が転送される前に効果を発揮する。3つのオプションのうち同一のPUTステートメントで指定することができるのは，PAGEとLINE だけである。あるPUTステートメントにPAGEとLINE が指定されている場合には，PAGEが先に適用される。

4. PAGEオプションが指定されていると，当該データ・セットの新しい「現在のページ」が定義される。「データ指定」も記入されている場合には，新しいページの定義が済んだ後で値が転送される。この後，PAGEオプションをもつPUTステートメントが実行されるか，PAGE形式項目が出て来るか，あるいは，ENDPAGE 条件が起って新しいページが定義されるまで，このページが「現在のページ」であるものとされる。新しい「現在のページ」は，最初の行（行1）を暗示している。

5. SKIPオプションが指定されていると，当該データ・セットの新しい「現在の行」が定義される。式も指定されている場合には，その式の値が整数(w)に変換される。PRINT 以外のファイルに関しては，wはゼロより大きくなければならない。データ・セットは，「現在の行」からw番目の行の先頭に位置設定される。式が指定されていない場合には，SKIP (1)であるとみなされる。

   PRINT ファイルに関しては，wはゼロ以下でもよい。wがゼロ以下の場合には，印刷位置が戻されるだけで行そのものは変らない。SKIP(w)が出されたとき「現在のページ」の残りの行がw行より少ない場合には，ENDPAGE条件が引き起される。

   注：　ENVIRONMENT属性のMEDIUMオプションで装置タイプとして3525ないし3525T と指定されているファイルに関しては，wはゼロより大きくなければならない。

6. LINEオプションが指定されていると，当該データ・セットの新しい「現在の行」が定義される。指定されている式の値は整数(w)に変換される。LINEオプションは，次の行が「現在のページ」のw番目の行となるようにブランク行を挿入することを指定するためのものである。「現在のページ」にすでにw行以上書き込まれている場合，またはwがOPENステートメントのPAGESIZE オプションでセットされている限界を越えている場合には，ENDPAGE 条件が引き起される。wがゼロ以下の場合には，wは1とみなされる。wが「現在の行」である場合は，ファイルの位置が最初の桁にセットされていない限り，ENDPAGE 条件が引き起される。ファイルが最初の桁に位置設定されているときは，SKIP(0)という指定と同じ結果になる。

   注：　ENVIRONMEMT属性のMEDIUMオプションで装置タイプとして3525Tと指定されているファイルに関しては，wは1か3でなければならない。

7. ファイルの開放の後の最初のPUT ステートメントで指定されている各種オプションの効果については，この節の「OPEN ステートメント」を参照のこと。

READ

READ ステートメントは，RECORD INPUT ファイルまたはRECORD UPDATE ファイルにあるレコードを変数または緩衝域へ転送するためのものである。

一般形式：

READ　オプション・リスト；

「オプション・リスト」の形式は図J.4 に示されている。

一般規則：

1. オプションを記入する順序は任意である。

2. FILEオプションは，読み取るべきレコードを収容しているファイルを指定するためのものである。このオプションは必ず指定しなければならない。ここで指定されているファイルがまだ開かれていない場合，このファイルがRECORD属性をもつものとして明示的に宣言されているならば，このファイルは暗示的に開かれる。これ以外の場合には，UNDEFINEDFILE条件が引き起される。

3. INTOオプションは，レコードを読み込むべき変数を指定するものである。変数が集合体である場合，変数は連続している記憶域になければならない。使い方によっては，UNALIGNED属性をもつ固定長ビット・ストリングの使用は禁じられている（詳しくは，「第12章：　レコード方式の転送」の「転送されるデータ」を参照のこと）。STRING以外のストリング擬似変数を指定してもよい。

4. KEYオプションあるいはKEYTOオプションは，KEYED ファイルに対してだけ指定することができる。

5. 当該ファイルがDIRECT属性をもつ場合には，KEYオプションを指定しなければならない。この「要素式」は，キーを表わす文字ストリングに変換される。このキーによって，どのレコードを読み取るかが決まる。

   INDEXED，SEQUENTIAL，およびKEYED 属性をもつファイルまたはキー順のVSAM編成で，SEQUENTIALならびにKEYED属性をもつファイルに対しても，KEYオプションを指定することができる。この場合，ファイルは，指定されているキーをもつレコードに位置設定される。この

```
                  ┌〔INTO（変数）〕 ┌KEY  （式）            ┐ 〔EVENT（事象変数）〕 ┐
                  │                └KEYTO（文字ストリング変数）┘                     │
FILE              │                                                                  │
 （ファイル名）    │〔SET（ポインター変数）〕 ┌KEY  （式）            ┐              │
                  │                        └KEYTO（文字ストリング変数）┘              │
                  └〔IGNORE（式）〕 〔EVENT（事象変数）〕                              ┘
```

図J.4 READステートメントのオプション・リストの形式

後，KEYオプションをもたないREAD ステートメントを用いて，それから先のレコードを順次方式で読み取ることができる。

6. 当該ファイルがSEQUENTIAL属性ならびにKEYED属性をもっている場合に限り，KEYTOオプションを指定することができる。このオプションは，読み取られているレコードのキーを，文字ストリング割当ての規則に従って，「文字ストリング変数」に割り当てるよう指定するためのものである。

KEYTOオプションでは，STRING以外であればどのストリング擬似変数でも指定することができるが，数字ピクチャー指定によって宣言されている変数を指定することはできない。文字ストリングの最大の長さは，256である。KEYTO変数への割当ては，つねに，INTO変数への割当ての後で行なわれる。値は次のようにして割り当てられる。

a. REGIONAL(1)： KEYTO オプションで指定されている文字ストリング変数が固定長である場合には，この文字ストリング変数について宣言されている長さに等しくなるよう，8文字の区域番号を左端でパディングまたは切捨てをして得た値が割り当てられる。文字ストリング変数が可変長である場合には，区域番号の先行ゼロが取り除かれて，その有効数字だけが割り当てられる。

b. REGIONAL(3), INDEXED，キー順VSAM： 文字ストリング変数について宣言されている長さに等しくなるよう，記録されたキーの右端でパディングまたは切捨てをして得た値が割り当てられる。

このようなパディングや切捨てに対しては，KEY条件は引き起されない。

7. EVENTオプションを指定すると，レコードが読み取られている間またはスキップされている間にも，処理を続けることができる。VSAM編成のSEQUENTIAL BUFFEREDファイルやDIRECT BUFFEREDファイルに対しては，このオプションを指定してはならない。

EVENTオプションをもつREADステートメントに制御が渡ると，「事象変数」が活用状態になり（すなわち，この変数を他の事象と結び付けることができなくなる），この変数に完了値'0'Bが与えられる。ただし，暗示的なファイルの開放によりUNDEFINEDFILE条件が引き起された場合にはこの限りでない（下の「注」を参照のこと）。この事象変数を指定しているWAITステートメントに制御が渡るまで，この事象変数は活用状態になっており，'0'Bという完了値を保持している。この事象変数を指定しているWAITステートメントに制御が渡ると，次のいずれかの措置がとられる。

a. READステートメントが正しく実行されて，このREADの結果としてENDFILE, TRANSMIT, KEY，あるいはRECORD条件のいずれも引き起されていない場合には，事象変数が「完了」にセットされ（完了値'1'Bが与えられ），事象変数が非活用状態になる（この事象変数を他の事象と結び付けることができるようになる）。

b. READステートメントを実行している間にENDFILE, TRANSMIT, KEY，あるいはRECORD条件が引き起された場合には，上述のWAITステートメントが出て来るまでは，これら各々の条件に起因する割込みは起らない。TRANSMITおよびRECORD条件だけが同時に起こる可能性がある点に注意すること。最後のオン・ユニットから戻ると，あるいは，いずれかのオン・ユニットがGO TOステートメントによって停止させられると，事象変数に完了値'1'Bが与えられ，非活用状態に切り替えられる。

注： READステートメントによって暗示的なファイルの開放が行なわれ，その結果UNDEFINEDFILE条件が引き起された場合には，この条件のためのオン・ユニットに直ちに入るが，事象変数は何も変化しない。す

なわち，事象変数は依然非活用状態にとどまり，READ ステートメントが出て来たとき保持していた値を引続き保持する。オン・ユニットでUNDEFINEDFILE 条件が訂正されなかった場合には，オン・ユニットから正常に戻った後でERROR 条件が引き起される。オン・ユニットでこの条件が訂正された場合（ファイルが正しく開かれた場合）には，オン・ユニットから正常に戻った後，事象変数が'0'B にセットされ，活用状態に切り替えられる。この後，READ ステートメントの実行が続行される。

8. SETオプションは，レコードを緩衝域へ読み込むよう指定すると共に，指定されているロケーター変数にポインター値を割り当てるよう指定するものである。ポインター値は，緩衝域の中のレコードを識別する。

9. SEQUENTIAL INPUT ファイルおよびSEQUENTIAL UPDATEファイルに対して，IGNOREオプションを指定することができる。IGNOREオプションで指定されている式の値が計算され，その値が整数に変換される。この整数(n)がゼロより大きい場合には，n個のレコードが無視される。当該ファイルを対象とするREAD ステートメントがこの後出て来ると，(n+1)番目のレコードがアクセスされる。nが1より小さい場合には，このオプションは何の効果ももたない。INTOオプション，SETオプション，あるいはIGNOREオプションをもたないREAD ステートメントは，IGNORE(1)をもつREADステートメントと同じである。

10. INDEXED 属性をもつファイルまたはキー順VSAM編成のファイルが順次方式でアクセスされている場合は，KEYオプションをもつREAD ステートメントを出すことによってそのファイルの位置を設定することができる。指定されているキーは，所要のレコードを識別するのに使用される。KEYオプションをもたないREAD ステートメントを用いて，以降のレコードを順次方式で読み取ることができる。これは，INPUTファイルの場合もUPDATE ファイルの場合も同様である。

次のような形式の2つの位置設定ステートメントを使用することができる。

```
READ FILE (ファイル式) INTO
     (変数) KEY (式);
READ FILE (ファイル式)
        SET (ポインター変数)
        KEY (式);
```

RETURN

RETURNステートメントは，このステートメントが入っているプロシージャーの実行を終了させるためのものである。RETURN ステートメントは，呼出しプロシージャーに制御を戻す。RETURN ステートメントが値を返すこともある。

一般形式：

```
オプション1：
   RETURN;
オプション2：
   RETURN (要素式);
```

一般規則：

1. プロシージャーが関数プロシージャーとして呼び出されているのでない場合は，このプロシージャーを終了させるためにはオプション1のRETURN ステートメントしか使用することができない。この場合，論理的に呼出しの後の場所に制御が戻される。

2. 関数プロシージャーとして呼び出されたプロシージャーを終了させる場合にだけ，オプション2のRETURN ステートメントが使用される。この場合，呼出しの行なわれた場所に制御が戻される。また，関数参照に返される値は，指定されている式の値を（呼び出された入口点に対して宣言されている）属性に合致するよう変換したものである。この属性は入口点で明示的に宣言することができる。明示的に宣言されていなければ，このプロシージャーを呼び出すのに使用された入口定数の頭文字にもとづいて，暗示される属性がとられる。

3. プロシージャーの終りに相当するENDステートメントに制御が渡ると，このENDステートメントは，このプロシージャーのための（オプション1形式の）RETURN ステートメントとして取り扱われる。

REVERT

REVERT ステートメントは，関連する最新のONステートメントの効果を取り消すのに使用される。REVERT ステートメントは，このステートメントが入っているブロックの内部にあり，しかもこのステートメントと同じ呼出しで実行されたONステートメントに対してだけ効力をもつ。あるブロックにおいてREVERT ステートメントを実行すると，このステートメントで指定されている条件を対象とし，しかもこのブロックにおいて実行されているONステートメントの処置の指定が取り消され，その後，このブロックの呼出しの時点で効力を有していた処置の指定が再び効力をもつようになる。

一般形式：

```
REVERT 条件;
```

構文法細則：

「条件」は，「H節： ON条件」に記載されている任意の条件である。

一般規則：

REVERTステートメントの実行によって上述の効果が得られるのは，次の2条件が満足されている場合に限られている。(1)同じ条件を指定しておりしかも同じブロックの内部にあるONステートメントが，このブロックが活用化された後で，実行されている。(2)この間，他に同じREVERT ステートメントは実行されていない。このいずれかの条件が満たされない場合には，REVERT ステートメントはナル・ステートメントとして取り扱われる。

REWRITE

REWRITE ステートメントは，更新ファイルに対してだけ使用することができる。このステートメントは，データ・セットの中の既存のレコードを置き換えるためのものである。

一般形式：

```
REWRITE FILE（ファイル式）
  〔FROM （変数）〕
  〔KEY  （要素式）〕
  〔EVENT （事象変数）〕；
```

構文法細則：

1. オプションを記入する順序は，任意である。

2. 「ファイル式」は，書き直すべきレコードを収容しているファイルを指定するためのものである。このファイルは，UPDATE 属性をもつものでなければならない。

3. FROM オプションは，指定されているファイルの中の既存のレコードと入れ替わるレコードを表わす変数を指定するためのものである。この変数が集合体である場合には，この変数は連続している記憶域になければならない。使い方によっては，UNALIGNED属性をもつ固定長ビット・ストリングの使用は禁じられている（詳しくは，「第12章： レコード方式の転送」の「転送されるデータ」を参照のこと）。

一般規則：

1. 「ファイル式」で参照されているファイルが未だ開かれていない場合，このファイルが RECORD 属性および UPDATE 属性をもつものとして明示的に宣言されているならば，このファイルは暗示的に開かれる。これ以外の場合には，UNDEFINEDFILE 条件が引き起される。

2. 当該ファイルがDIRECT 属性をもつ場合には，KEYオプションを指定しなければならない。これ以外の場合にKEYオプションを指定することはできない。ここで指定されている「要素式」は，文字ストリングに変換される。この文字ストリングは原始キーであり，これによってどのレコードを書き直すべきであるかが決まる。

3. INDEXED 属性をもつSEQUENTIALファイルまたはキー順VSAM編成のSEQUENTIALファイルに関しては，内蔵されたキーは，もとの内蔵されたキーと同じものでなければならない。

4. DIRECT属性をもつUPDATEファイル，または，SEQUENTIAL属性とUNBUFFERED属性の両方をもつUPDATEファイルに対しては，FROM オプションを指定しなければならない。FROM オプションをもたないREWRITE ステートメントの効果は次の通りである。

   a. 最後のレコードがINTOオプションをもつREADステートメントによって読み取られた場合は，FROM オプションをもたないREWRITE ステートメントはデータ・セットの中のそのレコードに対しては何の影響も与えない。

   b. 最後のレコードがSET オプションをもつREAD ステートメントによって読み取られた場合は，そのSETオプションのポインター変数が指す緩衝域で行なわれた何らかの割当てによって，このレコードが更新される。レコードがブロック化レコードである場合には，そのブロックの中のいずれかのレコードに対してREWRITE ステートメントを出すと，ブロック全体が書き直される。したがって更新されるレコードに対してREWRITE ステートメントを出さなくても，このレコードは書き直されることもあるわけである。

5. EVENTオプションを指定すると，レコードの書直しが行なわれているときにも処理を続けることができる。SEQUENTIAL BUFFEREDファイルまたはVSAM編成のDIRECT BUFFEREDファイルに対して，このオプションを指定することはできない。

EVENTオプションをもつREWRITE ステートメントに制御が渡ると，事象変数が活用状態に切り替えられ（すなわちこの事象変数を他の事象と結び付けることができなくなる)，この事象変数に '0'Bという完了値が与えられる。ただし，ファイルの暗示的な開放によってUNDEFINEDFILE条件が引き起された場合には，この限りでない（下の「注」を参照のこと）。この事象変数を指定しているWAITステートメントに制御が渡るまでは，この事象変数は活用状態にとどまり，'0'Bという完了値を保持する。この事象変数を指定しているWAITステートメントに制御が渡ると，次のいずれかの措置がとられる。

   a. REWRITE ステートメントが正しく実行され，その結果としてTRANSMIT，KEY，あるいはRECORD条件が引き起されていない場合には，事象変数は「完了」にセットされ，（'1'Bという完了値を与えられ），非活用状態に切り替えられる（すなわち，この変数を他の事象と結び付けることができるようになる）。

b. REWRITE ステートメントの実行が原因になってTRANSMIT, KEY, あるいはRECORD 条件が引き起された場合には，上述のWAIT ステートメントが出て来るまでこれら各々の条件のための割込みは起らない。TRANSMIT条件とRECORD 条件だけが一緒に起こる可能性があるという点に注意すること。最後のオン・ユニットから戻ると，あるいは，いずれかのオン・ユニットがGO TO ステートメントによって終了させられると，事象変数に '1'B という完了値が与えられ，この変数の状態が非活用状態に切り替えられる。

注： REWRITE ステートメントの実行によってファイルが暗示的に開放され，これがもとでUNDEFINEDFILE 条件が引き起された場合には，直ちに，この条件のためのオン・ユニットに入る。ただし，事象変数は何も変化しない。すなわち，事象変数は依然非活用状態にとどまり，REWRITE ステートメントが出て来たとき保持していた値をそのままもっている。オン・ユニットでこの条件が訂正されなかった場合には，オン・ユニットから正常に戻った後ERROR条件が引き起される。オン・ユニットでこの条件が訂正された場合（ファイルが正しく開かれた場合）には，オン・ユニットから正常に戻った後，事象変数が '0'Bにセットされ，活用状態に切り替えられる。この後，REWRITE ステートメントの実行が続行される。

SIGNAL

SIGNAL ステートメントは，割込みの発生をシミュレートするためのものである。また，このステートメントを用いて，このステートメントで指定されている条件に対する現在の処置の指定をテストすることができる。

一般形式：

```
SIGNAL 条件;
```

構文法細則：

「条件」は「H節： ON条件」に記載されている任意の条件である。

一般規則：

1. SIGNALステートメントが実行されると，ここで指定されている条件が実際に引き起された場合と同じような措置がとられる。すなわち，逐次的な実行が中断され，ここで指定されている条件のための現在のオン・ユニットに制御が渡される。オン・ユニットの実行が完了すると，SIGNALステートメントの直ぐ後のステートメントに制御が渡される。

2. CONDITIONと言うオン条件が割込みを引き起こすのは，SIGNALステートメントにCONDITIONと指定されている場合だけである。

3. ここで指定されている条件が禁止状態にある場合には，割込みは起こらず，SIGNAL ステートメントはナル・ステートメントと同じになる。

4. ここで指定されている条件のためのオン・ユニットが現在ない場合には，この条件に対するシステムの標準処置がとられる。

STOP

STOP ステートメントは当該プログラムを直ちに終了させるためのものである。

一般形式：

```
STOP;
```

一般規則：

STOP ステートメントが実行されると，終了のための措置がとられる前にFINISH 条件が引き起される。FINISH のためのオン・ユニットからの正常な戻りの後，当該プログラムが終了する。

WAIT

あるブロックのある活用化においてWAIT ステートメントが実行されると，このステートメントで指定されているいくつかの事象が完了するまで，このブロックのこの活動化に関する制御はこのWAIT ステートメントに保持されることになる。

一般形式：

```
WAIT (事象〔,事象〕・・・)〔(要素式)〕;
```

構文法細則：

1. 各々の「事象」は，事象変数または事象変数だけで構成されている配列である。

一般規則：

1. WAIT ステートメントを実行している間，恐らくは，別々の時点で，リストで指定されている事象名の一部またはすべてに関して次の条件が満足されるまでは，

   COMPLETION(事象)='1'B

   このブロックのこの活用化に関する制御は，このWAIT ステートメントに止めて置かれる。

2. 任意選択の「要素式」が指定されていない場合には，リストの中のすべての事象名が上の条件を満たした後

始めて，次のステートメントに制御が渡される。

3. 任意選択の「要素式」が指定されている場合には，WAIT ステートメントが実行されるときにその式の値が計算され，整数に変換される。この整数は，当該ブロックの制御を WAIT ステートメントの次のステートメントに渡す前に，上の条件を満たさなければならない，リスト内の事象の数を指定するためのものである。もちろん，このWAIT ステートメントが原因でオン・ユニットに入り，このオン・ユニットが異常停止すれば，このWAIT ステートメントの次のステートメントに制御は渡らない。

   「要素式」の値がゼロあるいは負である場合には，WAIT ステートメントはナル・ステートメントとして取り扱われる。「要素式」の値がリスト内の事象名の数（n）より大きい場合には，その値はnとみなされる。WAIT ステートメントが配列事象名を参照している場合には，配列の各々の要素がこのnの数に入る。

4. WAIT ステートメントで指定されているある事象変数が，このWAIT ステートメントより先に開始されたある入出力操作と結び付けられている場合は，この入出力操作が完了したときに，一般規則1で示した条件が満たされる。この場合，WAIT ステートメントの実行は入出力操作の完了に必要な要件である。WAIT ステートメントより先にあるいは WAIT ステートメントを実行している間に，この入出力操作に関係するすべての転送が完了した場合には，WAIT ステートメントは次のような措置をとる。入出力条件を一切引き起す必要なしにこの転送が完了しているならば，事象変数が「完了」にセットされる。転送は完了しているが条件を引き起す必要があるという場合には，事象変数が「異常」にセットされ（すなわち，STATUS（事象名）＝1），所要のON条件がすべて引き起される。最後のオン・ユニットから戻ると，事象変数が「完了」にセットされる。

5. 種々の入出力事象に関するON条件が引き起される順序は，リストに記入されている事象名の順序とは無関係である。ある事象に関してあるON条件が引き起されると，WAIT ステートメントが終了する前にあるいは他の入出力条件が引き起される前に，この事象に関する他のすべての条件が引き起される。ただし，このようにして入ったオン・ユニットのいずれか1つから異常な戻りが行なわれた場合には，この限りでない。

6. WAIT ステートメントから入ったいずれかのオン・ユニットで異常な戻りが行なわれた場合は，対応する事象変数が「完了」にセットされ，WAIT ステートメントの実行が終了し，異常な戻りで指定されている場所に制御が渡される。

7. WAIT ステートメントのリストの中に入出力操作と結び付けられている事象名があり，（所定の数だけの事象が完了したためにあるいは異常な戻りのために）このWAIT ステートメントが終了するまでにそのいくつかがまだ「完了」にセットされていない場合は，これらの未完了事象は，これらの事象を参照する他のWAIT ステートメントが実行されるまでは，「完了」にセットされない。

WRITE

WRITE ステートメントは，内部記憶装置にある変数から OUTPUTファイルないしUPDATEファイルにレコードを転送するためのRECORD転送ステートメントである。

一般形式：

```
WRITE FILE （ファイル式）
   FROM（変数）
   〔KEYFROM（要素式）〕
   〔EVENT （事象変数）〕；
```

構文法細則：

1. オプションを記入する順序は任意である。

2. 「ファイル式」は，レコードを書き込むべきファイルを指定するためのものである。このファイルは，OUTPUT 属性をもつRECORDファイル，またはDIRECT属性と UPDATE属性をもつRECORDファイルでなければならない。

3. FROMオプションは，書き込むべきレコードを表わす変数を指定するためのものである。この変数が集合体である場合には，この変数は連続している記憶域になければならない。使い方によっては，UNALIGNED 属性をもつ固定長ビット・ストリングの使用は禁じられている（詳しくは，「第12章：レコード式転送」の「転送されるデータ」を参照のこと）。

一般規則：

1. ファイルが開かれていない場合，RECORD属性とOUTPUT属性またはRECORD属性とUPDATE属性が明示的に宣言されているならばこのファイルは明示的に開かれる。これ以外の場合にはすべて，UNDEFINEDFILE条件が引き起される。

2. KEYFROMオプションが指定されている場合，「要素式」は文字ストリングに変換される。この文字ストリングはデータ・セット内のレコードが書き込まれる場所の相対位置を表わす原始キーである。REGIONAL（3）データ・セッ

トおよびINDEXED データ・セットに対しては，KEYFROM オプションで，記録されたキーも指定する。このキーの長さは，ENVIRONMENT属性のKEYLENGTHオプションによって決まる。

3. EVENTオプションを指定すると，レコードを書き込んでいるときにも処理を続けることができる。SEQUENTIAL BUFFERED ファイルまたはVSAM 編成のDIRECT BUFFERED ファイルに対して，このオプションを指定することはできない。

EVENTオプションをもつWRITEステートメントに制御が渡ると，「事象変数」が活用化される（すなわち，この変数を他の事象と結び付けることができなくなる）。ただし，ファイルの暗示的な開放によってUNDEFINEDFILE 条件が引き起された場合にはこの限りでない（下の「注」を参照のこと）。この事象変数を指定しているWAIT ステートメントに制御が渡るまでは，この事象変数は活用状態にとどまる。この事象変数を指定しているWAIT ステートメントに制御が渡ると，次のいずれかの措置がとられる。

a. WRITE ステートメントが所期通りに実行されておりこのWRITE ステートメントの結果としてTRANSMIT KEY，あるいはRECORD 条件が引き起されていない場合には，事象変数が「完了」にセットされ，その状態が非活用状態に切り替えられる（すなわち，この変数を他の事象と結び付けることができるようになる）。

b. WRITE ステートメントの実行の結果，TRANSMIT，KEY，あるいはRECORD条件が引き起された場合には，上述のWAIT ステートメントが出て来るまではこれら各々の条件のための割込みは起こらない。TRANSMIT 条件とRECORD 条件だけが一緒に引き起される可能性があるという点に注意すること。最後のオン・ユニットから戻ると，あるいは，いずれかのオンユニットがGO TO ステートメントによって終了させられると，事象変数は「完了」にセットされ，その状態が非活用状態に切り替えられる。

注： WRITEステートメントを実行した結果，ファイルが暗示的に開放され，そのためにUNDEFINEDFILE条件が引き起された場合には，直ちに，この条件のためのオン・ユニットに入る。ただし，事象変数は何も変化しない。すなわち，事象変数は依然非活用状態にとどまり，WRITEステートメントが出て来たときと同じ値を保持する。オン・ユニットでこの条件が訂正されなかった場合には，オン・ユニットから正常に戻った後，ERROR条件が引き起される。オン・ユニットでこの条件が訂正された場合（すなわち，ファイルが正しく開かれた場合）には，オン・ユニットから正常に戻った後，事象変数は，「未完了」にセットされ，さらに活用化される。この後，WRITEステートメントの実行が続行される。

プリプロセッサー・ステートメント

ここでは，プリプロセッサーの段階で実行することのできるすべてのステートメントを，アルファベット順に説明する。

%ACTIVATE

省略形： %ACT

%ACTIVATE ステートメントで識別名を指定すると，この識別名は活用状態に切り替えられ，置換え可能になる。したがって，この後，プリプロセッサー・ステートメント以外のステートメントでこの識別名が指定されていると，この識別名が活用状態にある限り，この識別名の置換えが開始される。

一般形式：

```
%〔ラベル：〕・・・ ACTIVATE識別名
 〔RESCAN|NORESCAN〕〔，識別名
 〔RESCAN|NORESCAN〕〕・・・；
```

構文法細則：

1. 各々の識別名は，プリプロセッサー変数，プリプロセッサー・プロシージャー名，SUBSTR組込み関数，LENGTH組込み関数，またはINDEX組込み関数でなければならない。

2. プリプロセッサー・プロシージャーの中に%ACTIVATE ステートメントを置くことはできない。

一般規則：

1. 識別名の状態を，当初，%ACTIVATE ステートメントによって活用化することはできない。これを行なうには，%DECLARE ステートメントでこの識別名を指定すればよい。%DEACTIVATEステートメントによって識別名を非活用状態にしたあとでだけ，%ACTIVATE ステートメントを用いてその識別名を活用化することができる。

2. ある識別名が活用状態にある（この識別名がプリプロセッサー変数であるならば，さらに，この識別名に値が与えられている）場合は，非プリプロセッサー・ステートメントの中にこの識別名が出て来ると，置換えが開始される。ただし，コメントの中または単一引用符の中にこの識別名が出て来た場合にはこの限りでない。たとえば，原始プログラムの中に次のような一連のステートメントがあるものとすれば，

```
%DECLARE I FIXED, T CHARACTER;
%DEACTIVATE I;
%I=15;
%T='A(I)';
S=I*T*3;
%I=I+5;
%ACTIVATE I;
%DEACTIVATE T;
R=I*T*2;
```

プリプロセッサーの処理が済んだとき，上記のステートメント群から生成されたテキストは次のようになる(置換えの結果，ブランクになった部分は示してない)。

```
S=I*A(I)*3;
R=20*T*2;
```

3. RESCANまたはNORESCANで参照されている識別名がFIXEDタイプのプリプロセッサー変数の名前である場合，またはFIXEDタイプの値を返すプリプロセッサー・プロシージャーの名前である場合は，いずれのオプションが指定されていても，出力の流れにおいて置換えが行なわれる。
RESCANまたはNORESCANで参照されている識別名がCHARACTERタイプのプリプロセッサー変数の名前である場合，またはCHARACTERタイプの値を返すプロシージャーの名前である場合は，それぞれのオプションは，次のような意味をもつ。

 a. RESCANは，プリプロセッサーによってこの識別名が走査されたとき，出力の流れにおいて置換えを通常通り行なうことを指定するものである。

 b. NORESCANは次のような指定を行なう。

  (1) プリプロセッサーによってこの識別名が走査されたとき，出力の流れにおいてこの識別名をこの識別名が表わしている変数の現在の値であるテキスト，またはこの識別名が表わしているプロシージャーを呼び出した結果であるテキストと置き換える。

  (2) さらに置換えを行なうためにこのテキストを再走査するようなことはしない。

いずれのオプションも指定されていない場合には，RESCANの指定があるものとみなされる。

4. すでに活用状態にあるプリプロセッサー識別名を対象とする%ACTIVATEステートメントを実行しても何も行なわれない。

%割当てステートメント

%割当てステートメントは，プリプロセッサー式の値を計算し，その値をプリプロセッサー変数に割り当てるのに使用される。

一般形式：

```
%〔ラベル：〕・・・ プリプロセッサー変数=
                   プリプロセッサー式；
```

一般規則：

プリプロセッサー変数に割り当てられる値が文字ストリングである場合，この文字ストリングにプリプロセッサー・ステートメントが含まれていてはならない。

%DEACTIVATE

省略形：　%DEACT

%DEACTIVATEステートメントで識別名を指定すると，この識別名は非活用状態に切り替えられ，置換え不可能な状態に置かれる。したがって，この後，非プリプロセッサー・ステートメントの中にこの識別名があっても，置換えは開始されない(もちろん，この間にこの識別名が再び活用化されていれば，この限りでない)。

一般形式：

```
%〔ラベル：〕・・・ DEACTIVATE識別名
            〔,識別名〕・・・；
```

構文法細則：

1. 各々の「識別名」は，プリプロセッサー変数，SUBSTR組込み関数，またはプリプロセッサー・プロシージャー名でなければならない。

2. プリプロセッサー・プロシージャーの中に%DEACTIVATEステートメントを置くことはできない。

一般規則：

ある識別名を非活用状態にしても，この識別名は値を失うわけではないし，後にプリプロセッサー・ステートメントの中でこの識別名に新しい値を与えるのが妨げられるわけでもない。非活用状態にするということは，単に，ある特定の識別名の置換えを阻止することであるにすぎない。すでに非活用状態にあるプリプロセッサー識別名を，非活用化しようとしても，何も行なわれない。

%DECLARE

省略形：　%DCL

%DECLAREステートメントは，識別名をプリプロセッサー変数ないしプリプロセッサー・プロシージャー名として認めさせるためのものであり，同時に，この識別名を活用状態に置く働きもする。ある識別名を，他のプリプロセッサー・ステートメントの中で変数またはプロシージャーとして使用するためには，前もって%DECLARE

ステートメントの中でこの識別名を宣言しておかなければならない。

一般形式：

```
%〔ラベル：〕・・・
  DECLARE 識別名
    {FIXED|CHARACTER|ENTRY|BUILTIN}
  〔，識別名
    {FIXED|CHARACTER|ENTRY|BUILTIN}〕・・・；
```

構文法細則：

1. 「識別名」がプリプロセッサー変数である場合には，CHARACTERまたはFIXEDと指定しなければならない。「識別名」がプリプロセッサー・プロシージャー名である場合には，必要に応じて，入口宣言を指定することができる。プリプロセッサー・プロシージャーの入口名を明示的に宣言するには，この入口名を%PROCEDUREステートメントのラベルとして指定しておけばよい。しかし，この明示的宣言では，プリプロセッサー・プロシージャー名は活用化されない。

2. %DECLARE ステートメントでは，「一般形式」で示した属性しか指定することができない。

3. 普通のDECLARE ステートメントの場合と同様，共通属性を因数化することができる。

4. 走査のときには，%DECLARE ステートメントのラベルは無視される。

一般規則：

1. CHARACTER属性で長さを指定することはできない。CHARACTERと指定されていると,その識別名は,最大の長さをもたない,長さの変わる文字ストリングを表わすものとみなされる。

2. FIXED属性をもつものとして宣言されているプリプロセッサー変数には，省略時属性としてDECIMAL と(5,0)という属性が与えられる。

3. プリプロセッサー宣言が走査されるまでは，その宣言が知られることはない。したがって，プリプロセッサー変数の宣言ないしプリプロセッサー・プロシージャーの宣言が走査される前に，その変数ないしプロシージャーへの参照が出て来ると，この参照は誤りになる。

4. あらゆるプリプロセッサー変数，プリプロセッサー・プロシージャー名，およびプリプロセッサー・ラベルの有効範囲は，当該プリプロセッサーによって走査される原始プログラム全体である(この識別名を宣言し直しているプリプロセッサー・プロシージャーは除く)。プリプロセッサー・プロシージャー内の宣言の有効範囲は，そのプロシージャーだけである。

5. 原始プログラムの中にある1つのプリプロセッサー・プロシージャーについて1つずつ入口宣言を指定することができる。入口宣言は，入口名を活用化するのに使用される。プリプロセッサー関数が呼び出されるたびに，その引数は，必要に応じて，対応するパラメーターの属性に合致するよう変換される。

   呼出しの際の引数とパラメーターとの関係については，第I部の「コンパイル時の諸機能」の「プリプロセッサー・プロシージャー」を参照のこと。

6. プリプロセッサーの%DECLARE ステートメントが出て来ると，このステートメントは%ACTIVATEステートメントとして働き，RESCAN オプションと併用することによって，このステートメントで識別されているすべてのプリプロセッサー変数を活用化する。

7. SUBSTR，LENGTH，またはINDEXに対してだけBUILTIN 属性を指定することができる。この属性は，この属性をもつ識別名がこれと同じ名前の組込み関数であることを指定するためのものである。

%DO

%DOステートメントは，%ENDステートメントと対になって，ある1つのプリプロセッサーDOグループの境界を指定するのに使用される。%DOステートメントをこれ以外の目的で使用することはできない。

一般形式：

```
%〔ラベル：〕・・・DO [i=m [TO m2〔EY m3〕]]
                           [BY m3〔TO M2〕]
```

構文法細則：

「i」はプリプロセッサー変数を表わす。「m1」，「m2」，「m3」はプリプロセッサー式である。

一般規則：

プリプロセッサーDOグループの展開は，これと対応する非プリプロセッサーDOグループの展開と同じであり，「i」，「m1」，「m2」，および「m3」の意味は，非プリプロセッサーDOグループのこれらに対応する式の意味と同じである。

このステートメントの説明と例については，「第15章：コンパイル時の諸機能」の「プリプロセッサーDOグループ」を参照のこと。

%END

%ENDステートメントは，%DOステートメントまたは%PROCEDUREステートメントと対になって，プリプロセッサーDOグループないしプリプロセッサー・プロシージャーの境界を指定するのに使用される。

一般形式：

%〔ラベル：〕・・・ END〔ラベル〕；

構文法細則：

ENDの後のラベルは，%PROCEDUREステートメントまたは%DOステートメントのラベルでなければならない。複数終止を行なうこともできる。

%GO TO

省略形： %GOTO

%GO TOステートメントがあると，プリプロセッサーは，指定されているラベルの所から走査を継続する。

一般形式：

%〔ラベル：〕・・・ GO TO ラベル；

一般規則：

1. GO TOというキーワードの後にあるラベルによって走査が移行する地点が決まる。このラベルはプリプロセッサー・ステートメントのラベルでなければならない。ただし，プリプロセッサー・プロシージャーのラベルであってはならない。

2. プリプロセッサーGO TOステートメントは，このステートメント自体が入っているプリプロセッサー・プロシージャーの外部の地点に制御を渡すことはできない。いいかえれば，キーワードGO TOの後のラベルは，このプロシージャーの中に含まれているものでなければならない。

3. 組み入れられるテキストの中で%GO TOステートメントを使用する上での制約については，「%INCLUDE ステートメント」を参照のこと。

%IF

%IFステートメントは，あるプリプロセッサー式の値をもとにして，走査の流れを切り替えることができる。

一般形式：

%〔ラベル：〕・・・IFプリプロセッサー式
　%THEN プリプロセッサー文節－1
　〔%ELSE プリプロセッサー文節－2〕

構文法細則：

プリプロセッサー文節は，%DECLARE，%PROCEDURE，%END，%DO（パーセント記号も含めて），または，プリプロセッサーDOグループ（パーセント記号も含めて），以外の1個のプリプロセッサー・ステートメントから構成される。これ以外の構文法は，非プリプロセッサーIFステートメントの構文法と同じである。

一般規則：

1. プリプロセッサー式の値が計算され，ビット・ストリングに変換される（ビット・ストリングへの変換ができなければエラーになる）。このストリングの中に値1をもつビットがある場合には，文節1が実行され，文節2は指定されていても無視される。ビットがすべて0である場合には，文節1は無視され，文節2が指定されていればこの文節が実行される。いずれの場合にも，いずれかの文節にある%GO TOステートメントによって別の場所から走査が続行されるのでない限り，IFのすぐ後のステートメントから走査が続行される。

2. %IFステートメントをネストすることができる。このネスティングに関する規則は，非プリプロセッサーIFステートメントのネスティングに関する規則と同じである。

%INCLUDE

%INCLUDEステートメントは，現在走査されている原始プログラムに，ストリング形式の外部テキストを組み入れるのに使用される。組み入れられたテキストは，プリプロセッサーによる処理の済んだ後のテキストの一部となる。

一般形式：

当コンパイラーにおいては，%INCLUDE ステートメントは次のように定義される。

%〔ラベル：〕・・・ INCLUDE
　｛識別名－1（識別名－2）｜識別名－2｝
　〔，識別名－1（識別名－2）｜，識別名－2〕

構文法細則：

1. 「識別名－1」は，原始ステートメント・ライブラリーの中のサブライブラリーを表わす。「識別名－2」は，ブック名を表わす。これらの識別名によって，原始プログラムに組み入れるべき外部テキストを識別する。

2. 「識別名－1」が記入されていない場合は，原始ステートメント・ライブラリーのPL/Iサブライブラリーが指定されているものとみなされる。

3. プリプロセッサー・プロシージャーの中で %INCLUDEス

テートメントを使用することはできない。

一般規則：

1. 組み入れられるテキストに，非プリプロセッサー・ステートメントおよび／またはプリプロセッサー・ステートメントが含まれていてもよい。組み入れられるテキストにおける論理レコードの許容される最大の長さは，８０バイトである。
2. 組み入れられるテキストは，原始プログラムと同様，逐次，走査される。すなわち，プリプロセッサー・ステートメントが実行され，必要に応じて，置換えが行なわれる。
3. ％INCLUDE ステートメントをネストすることができる。いいかえれば，組み入れられるテキストに％INCLUDE ステートメントが含まれていてもよい。組み入れられたテキストの中で％GO TO ステートメントを使用して，原始プログラムの中のある場所に制御を渡すこともできるし，ネスティングの外側のレベルにある組み入れられたテキストの中の場所に制御を渡すこともできる。しかし，この逆は許されない。これに似た関係は実行の反復を指定しているDOグループのネスティングにおいても見られる。すなわち，ある内側のDOグループからこのグループを含んでいる外側のDOグループに制御を渡すことはできるけれども，外側のグループからこのグループに含まれている内側のグループに制御を渡すことはできない。ネスティングの許容される最大の深さは，４９である。
4. 組み入れられたテキストの中のプリプロセッサー・ステートメントは，完全な形になっていなければならない。たとえば，ある％IFステートメントの半分が組み入れられたテキストの中にあり，残り半分が原始プログラムの他の部分にあるというのは許されない。

例：

原始プログラムの中に次のような個所があるものとする。

```
%DECLARE (FILENAME1, FILENAME2)
            CHARACTER;
 % FILENAME1 = 'MASTER';
 % FILENAME2 = 'NEWFILE';
 % INCLUDE DCLS;
```

また，原始ステートメント・ライブラリーのPL／I サブライブラリーの中にあるブックDCLS に，次のステートメントが含まれているものとする。

```
DECLARE (FILENAME1, FILENAME2)
        FILE RECORD INPUT
        DIRECT KEYED;
```

この場合，プリプロセッサーによる処理を経た後のテキストには次のステートメントが組み込まれる。

```
DECLARE (MASTER, NEWFILE)
        FILE RECORD INPUT
        DIRECT KEYED;
```

これがファイル宣言の中央ライブラリーを使用する方法である点に注意すること。ユーザーは，宣言されるファイルに独自の名前を与えて使用する。

％ナル・ステートメント

％ナル・ステートメントの用途の１つは，％GO TO ステートメントによる制御の移行先を設置することである。また，ネストされた％IFステートメントのELSE クローズの釣合いをとるためにこのステートメントを使用すると都合がよい。

一般形式：

```
%〔ラベル：〕・・・;
```

％PROCEDURE

省略形：　％PROC

％PROCEDUREステートメントは，％END ステートメントと対になって，ある１つのプリプロセッサー・プロシージャーの境界を指定するのに使用される。このようなプリプロセッサー・プロシージャーは，プリプロセッサーの段階でしか実行することのできない内部関数プロシージャーである。

一般形式：

```
%ラベル： 〔ラベル：〕・・・ PROCEDURE
    〔（識別名〔，識別名〕・・・）〕
    RETURNS{CHARACTER|FIXED};
```

構文法細則：

1. 各々の「識別名」は関数プロシージャーのパラメーターである。最高１５の「識別名」を指定することができる。
2. RETURNS の属性のリストの中でCHARACTER 属性またはFIXED属性を指定して，当該関数プロシージャーから返される値のタイプを指示しなければならない。これに省略時属性が適用されることはない。

一般規則：

1. プリプロセッサー・プロシージャーの中では，次にあげるステートメントおよびグループしか使用することができない。
   a．プリプロセッサー割当てステートメント
   b．プリプロセッサーDECLARE ステートメント
   c．プリプロセッサーDOグループ
   d．プリプロセッサーGO TO ステートメント
   e．プリプロセッサーIFステートメント
   f．プリプロセッサー・ナル・ステートメント
   g．プリプロセッサーRETURN ステートメント

これらのステートメントおよびDOグループはすべて，これまでに説明して来たプリプロセッサー・ステートメントおよびDOグループの構文法と一般規則に従っていなければならない。ただし，いずれの場合も，パーセントの記号をとり除かなければならない。

2. プリプロセッサー・プロシージャーの中のGO TO ステートメントによって，このプロシージャーの外部の地点に制御を渡すことはできない。

3. 一般規則1の記述からもわかる通り，プリプロセッサー・プロシージャーをネストすることはできない。

4. プリプロセッサー・プロシージャーは，プリプロセッサー・ステートメントの中の関数参照によって呼び出すことができる。また，関数プロシージャー名が活用状態にある場合には，非プリプロセッサー・ステートメントの中にその名前が出て来ると，プリプロセッサー・プロシージャーが呼び出される。

プリプロセッサーRETURNステートメント

プリプロセッサーRETURNステートメントは，プリプロセッサープロシージャーの中でしか使用することができない。このため，プリプロセッサーRETURNステートメントの冒頭に%をつけることはできない。プリプロセッサーRETURNステートメントは，当該プリプロセッサー・プロシージャーの呼出し点に値ならびに制御を返すためのものである。

一般形式：

〔ラベル：〕・・・ RETURN
(プリプロセッサー式)；

一般規則：

プリプロセッサー式の値が，%PROCEDUREステートメントで指定されているRETURNSの属性のリストの属性に変換され，この後，この値が呼出し点に返される。呼出し点が非プリプロセッサー・ステートメントの中にある場合には，返された値が関数参照と置き換えられた後，プリプロセッサーによってこの値を置き換えることができる。

プリプロセッサー式に関する規則では，プリプロセッサー・プロシージャーから返された値にプリプロセッサー・ステートメントが含まれることは許されていないので，注意すること。

リスティング制御ステートメント

%PAGE

%PAGEステートメントの後に記入されているステートメントは，プログラム・リストの中で次のページの最初の行に印刷される。

一般形式：

%PAGE；

構文法細則：

1. 用紙送りが行なわれる関係上，このステートメントと同じ行に他のステートメントがあってはならない。

2. このステートメントが注釈の中または他のステートメントの中にある場合には，このステートメントは出力形式に何の効果も及ぼさない。

一般規則：

1. プリプロセッサーを使用する場合も使用しない場合も，このステートメントを使用することができる。このステートメントは前原始リストと原始リストの双方に対して効力をもつ。

2. %PAGEステートメントの指示が実施された後，このステートメントは，プリプロセッサーによって印刷されることなく，コンパイラーによってテキストから削除される。すなわち，このステートメントは，リストには現われない。%PAGEステートメントに注釈または注釈の一部があっても，これも削除され，リストには現われない。

3. プリプロセッサーを使用する場合，%PAGE ステートメントが他の1つ以上のステートメントと同じ行に記入されていると，プリプロセッサーは%PAGEステートメントを移動して，コンパイラーに渡されるテキストの中ではこのステートメントが他のステートメントのない行に来るようにする。したがって，前原始リストでは用紙送りは行なわれないが，原始リストでは行なわれる。

4. プリプロセッサーを使用する場合，%PAGEステートメントがある行の末尾に，次の行にまたがる形で識別名が記入されていると，この識別名の2つの部分は連結されて1つの語になる。%PAGEステートメントのある行に充分なスペースがあれば，この識別名の後の部分が前の部分のある行に移動される。スペースがなければ，この識別名は1つに連結されて新しい行の冒頭に印刷される。

%SKIP

プログラム・リストにおいて，%SKIPステートメントの後は，指定の行数だけブランクになる。

一般形式：

%SKIP〔(n)〕；

構文法細則：

1. このステートメントの指示が実施に移されるためには，このステートメントは，他のステートメントが記入されていな

い行になければならない。

2. このステートメントが注釈または他のステートメントの中にある場合には，このステートメントは出力形式に何の効果も及ぼさない。

3. nは，1～999の範囲の10進整数定数でなければならない。nが記入されていない場合，nは1とみなされる。

一般規則：

1. プリプロセッサーを使用する場合も使用しない場合も，このステートメントを使用することができる。このステートメントは前原始リストおよび原始リストの双方に対して効力をもつ。

2. %SKIPステートメントの指示が実施に移された後，このステートメントは，プリプロセッサーによって印刷されることなく，コンパイラーによってテキストから削除される。すなわち，このステートメントはリストには現われない。
%SKIPステートメントのある行に注釈または注釈の一部が記入されていても，これは無視され，リストには現われない。

3. プリプロセッサーを使用する場合，%SKIPステートメントが他の1つ以上のステートメントと同じ行に記入されていると，プリプロセッサーは%SKIPステートメントを移動して，コンパイラーに渡されるテキストではこのステートメントが他のステートメントのない行に来るようにする。したがって，前原始リストではこのステートメントの効果は現われないが，原始リストではその効果が現われる。

4. プリプロセッサーを使用する場合，%SKIPステートメントのある行の末尾に次の行にまたがる形で識別名が記入されていると，この識別名の2つの部分は，連結されて1つの単語になる。前の部分が記入されている行にスペースが充分ある場合には，後の部分がその行に移される。スペースがなければ，連結されて1つになった単語が新しい行の冒頭に印刷される。

5. nが当該ページの残りの行数より大きい場合には，%SKIPステートメントのかわりに，%PAGEステートメントに相当するものが実行される。

K節： データ・マップの作成

この節では，当コンパイラーのもとで行なわれる構造のマップ作成と緩衝域内のレコードの境界合せについて説明する。プログラムを書くときに，このようなことを心に留めて置かなければならない場合もあるので，本書でもこれらの事項をとり上げて説明する。しかし，ストリーム式転送を採用しているプログラマーあるいはUNALIGNED 属性をもつ（ビット・ストリング以外の）データを取り扱うプログラマーにとっては，以下の説明は不要である。以下の説明は，ALIGNED 属性をもつ構造を取り扱いレコード式転送（特に，位置指定方式）を採用しているプログラマーに対するものである。

構造のマップ作成

大構造でも小構造でも，長さ，所要の境界合せ，およびダブルワード境界を基準とする相対位置は，そのメンバーの長さ，所要の境界合せ，および相対位置によって決まる。各レベルごとに順次これらの所要事項を決定し最後に構造全体のこれらの所要事項を決定する過程を構造のマップ作成と呼んでいる。

構造のマップを作成するとき，コンパイラーは，構造のメンバーとメンバーの間の使用されていない記憶域（パディング）を最小にする。コンパイラーは，構造が割り振られる前に，（事実上）以下で説明する規則に従って，構造のマップ作成を完了する。レコード式入出力を使用する場合にある構造に必要なレコードの長さを決めたり，位置指定方式の入出力において構造の境界合せを正しく行なわせるのに必要なパディングや再編成の量を決めたりするにあたっては，ユーザーは以下の規則を理解しておかなければならない（位置指定方式の入出力に関しては，この節の「レコードの境界合せ」を参照のこと）。

構造のマップ作成とは，物理的な操作を行なうものではない。以下の説明の中で「移す」とか「オフセット」とかの単語が使われているが，これらの単語は単に説明のためだけのものであり，記憶域の中で実際に移動が行なわれることを意味しているわけではない。構造の割振りが行なわれるときには，マップ作成の結果として，相対位置はすでにわかっている。

ある構造全体のマップを作成すると，項目（要素，または，内部における個々のマップがすでに出来上っている小構造）は，順次，接合されて対になってゆく。一旦ある対ができ上ると，この対はまた別の単位と対になる1単位となり，こうした操作が繰り返されて構造全体のマップが出来上るわけである。この説明からもわかるように，マップ作成の規則は次の2つに分類することができる。

対を作る順序に関する規則

一対のマップの作成に関する規則

以下でこれらの規則について説明する。また，この節の後の方では，例を上げてこれらの規則の適用について詳しく説明する。

注： これらの規則を理解するには，論理レベルとレベル番号との相違を正しく認識しなければならない。一番大きいレベル番号をもつ項目が，必ずしも一番下の論理レベルの項目であるわけではない。構造の宣言で指定されているレベル番号が首尾一貫している場合や，適当に字下げされている場合（後の方で取り上げる例を参照のこと）は，論理レベルは一目瞭然である。いずれにせよ，構造の各々の項目の論理レベルは，構造の宣言の最初から各々の項目に順次次の規則を適用することによって決定される。

> ある項目の論理レベルは，つねに，直接この項目を含んでいる構造の論理レベルより1つ下である。

```
DCL   1A，4B，5C，5D，3E，8F，7G
        1   2   3   3   2   3   3
```

> 下の行の数字は，この宣言における各々の項目の論理レベルを示している。

対を作る順序に関する規則

対を作る順序は，次のようにして決定される。

1. 論理レベルが一番下の小構造を探す（この論理レベルをnとする）。
2. 論理レベルnの小構造の数が2以上の場合には，宣言の中で最初に出て来た論理レベルnの小構造をとる。
3. 一対のマップの作成に関する規則（下記）に従って，この小構造の中の最初の2つの要素を対にする。この対が1つの単位となる。

4. この単位と，（この小構造の宣言の中に次の要素があれば）次の要素とで対を作り，より大きな単位を構成する。

5. この小構造の中のすべての要素が1つの単位にまとめられるまで，上の規則4を繰り返す。これが済むと，この小構造のマップが完成したことになる。ここで，この小構造の所要の境界合せと長さ（パディングを含む）が決まる。これ以後は，プログラマーの側でこの構造の宣言を変更しない限り，これらが変化することはない。この小構造のダブルワード境界からのオフセットもすでに決められている。このオフセットは，この小構造を含んでいる構造のマップを作成するときに重要な意味をもち，このようなマップ作成の結果変化することもあるという点に注意すること。

6. 宣言の中で論理レベルnにある，次の小構造に対して（このような小構造があれば），上の規則3～5を繰り返す。

7. 論理レベルnの小構造のマップがすべて完成するまで，規則6を繰り返す。これが済むと，構造のマップという観点から見る限り，これら各々の小構造を1つの要素とみなすことができる。

8. 次の論理レベルの小構造に対して，これまでの操作を繰り返す。すなわち，nを（n－1）と置き，規則2～7を繰り返す。

9. n＝1になるまで規則8を繰り返す。次いで，大構造に対して規則3～5を繰り返す。

一対のマップの作成に関する規則

（すでに述べた通り，ここでは明らかに物理的な記憶装置を指す用語を使用しているが，これは単に説明の便宜を図るためのものである。これらの用語が意味している記憶装置は，ダブルワードが数多く並んでいる想像上のモデルであると考えてもよい。各々のダブルワードには0～7の番号をもつ8つのバイトがあると考えれば，ダブルワード境界からのオフセットを表わすことができる。さらに，このモデルのバイトには，任意のバイトをゼロとして始まる一連番号もついていると考えれば，構造の始まりからの長さとオフセットを表わすことができる。）

1. 対の最初の項目をダブルワード境界に揃える。あるいは，この項目がすでにマップの出来上った小構造である場合には，ダブルワード境界から指示されている距離だけ離れた位置にこの項目を置く。

2. 対のもう一方の項目を，最初の項目の後にある最初の有効な位置に揃える。この位置は，2番目の項目の要求する境界合せによって決まる。種々の要素が要求する境界合せと長さを図K.1と図K.2に示す。（この項目が小構造である場合には，これに必要な境界合せはすでに決定されている。）

3. 最初の項目に必要な境界合せが許す限り，最初の項目を2番目の項目の方にずらせる。こうしてずらせた距離が，この対のダブルワード境界からのオフセットである。

上述の操作が完了すると，2つの項目の間のパディングが最小化されたことになり，このパディングはこれ以後の操作において変化することはない。この時点で，この対は，固定長で，所要の境界合せをもつ1つの単位と考えることができるようになる。すなわち，この対の長さは，2つの項目の長さとパディングの和であり，所要の境界合せとは，（2つの項目の境界合せが異なる場合には）大きい方の境界合せである。

UNALIGNED属性の効果

以下で取り上げる構造のマップ作成の例は，ALIGNED 属性をもつ構造にマップ作成の規則が適用される状況を示すためのものである。異なる境界合せが数多くあれば，ALIGNED 属性をもつ構造のマップ作成の方がより複雑であるため，このような例をとり上げた。一般に，UNALIGNED属性を適用すると，フルワードの境界合せやダブルワードの境界合せはバイトの境界合せに下げられ，ビット・ストリングの境界合せはバイトからビットに下げられる。構造のマップ作成に関する規則はいずれの場合にも同じであるが，UNALIGNED属性をもつ構造においては境界合せの所要条件が下げられるわけである。したがって，項目と項目の間の使用されない記憶域は，バイト内のビット・パディングだけであって，バイトパディングになることはない。構造にビット・ストリングが含まれている場合には，ビット・パディングが行なわれる。

EVENTデータやAREAデータには，UNALIGNED 属性を与えることができない。UNALIGNED 属性をもつ構造の中にこのようにUNALIGNED 属性をもち得ない要素が含まれている場合には，この要素についてはUNALIGNED属性が無視される。すなわち，コンパイラーはこの要素をALIGNED属性をもつものとして処理した後，エラー・メッセージを印刷する。たとえば，あるプログラムの中に次のような宣言があるものとする。

```
DECLARE 1 A UNALIGNED,
        2 B,
        2 C AREA(100);
```

この場合，Cが受け継いだ属性UNALIGNEDは，AREA 属性と両立し得ないので，CにはALIGNED 属性が与えられる。

<table>
<tr><th>変数のタイプ</th><th>計算機内部では
次のような形で貯えられる</th><th>所要記憶域の大きさ
(バイト)</th><th>所要の
境界合せ</th><th>注　釈</th></tr>
<tr><td>BIT(n)</td><td>8ビットのグループごとに1バイト(またはその一部)</td><td>CEIL(n/8)</td><td rowspan="4">バイト</td><td rowspan="4">データは0～7の間の任意のバイトから始まることができる</td></tr>
<tr><td>CHARACTER(n)</td><td>1文字につき1バイト</td><td>n</td></tr>
<tr><td>PICTURE</td><td>各PICTURE 文字ごとに1バイィ(ただし, VとKを除く)</td><td>VとKを除くPICTURE 文字の数</td></tr>
<tr><td>DECIMAL FIXED(p,q)</td><td>パック10進数形式
(数字1つにつき1/2 バイト+符号のための1/2 バイト)</td><td>CEIL((P+1)/2)</td></tr>
<tr><td>BIT(n)VARYING</td><td>2バイトの接頭語をもつBIT(n)と同じ</td><td>2+CEIL(n/8)</td><td rowspan="3">ハーフワード</td><td rowspan="3">データが始まるバイトは0,2,4,または6のバイトだけである</td></tr>
<tr><td>CHARACTER(n)
VARYING</td><td>2バイトの接頭語をもつCHARACTER(n)と同じ</td><td>2+n</td></tr>
<tr><td rowspan="2">BINARY FIXED(p,q)
p<=15

p> 15</td><td>ハーフワードの
2進整数</td><td>2</td></tr>
<tr><td>フルワードの
2進整数</td><td rowspan="6">4</td><td rowspan="9">フルワード</td><td rowspan="9">データが始まるバイトは0または4のバイトだけである</td></tr>
<tr><td>BINARY FLOAT(p)
p< 22</td><td rowspan="2">短精度
浮動小数点数</td></tr>
<tr><td>DECIMAL FLOAT(p)
p<7</td></tr>
<tr><td>POINTER</td><td>—</td></tr>
<tr><td>OFFSET</td><td>—</td></tr>
<tr><td>FILE</td><td>—</td></tr>
<tr><td>ENTRY</td><td>—</td><td rowspan="2">8</td></tr>
<tr><td>LABEL</td><td>—</td></tr>
<tr><td>EVENT</td><td>—</td><td>32</td></tr>
<tr><td>BINARY FLOAT(p)
21<p<54</td><td rowspan="2">長精度
浮動小数点数</td><td rowspan="2">8</td><td rowspan="3">ダブルワード</td><td rowspan="3">データが始まるバイトはバイト0だけである</td></tr>
<tr><td>DECIMAL FLOAT(p)
6<p<17</td></tr>
<tr><td>AREA</td><td>—</td><td>16+エリアの大きさ</td></tr>
</table>

図K.1　ALIGNED データの境界合せの要件

構造のマップ作成の例

この例は，以下のように宣言されている構造に対して，マップ作成の規則がどう適用されるかを示すものである。

```
DECLARE 1 A ALIGNED,
          2 B POINTER,
          2 C,
            3 D FLOAT DECIMAL(14),
            3 E,
              4 F LABEL,
              4 G,
                5 H CHARACTER(2),
                5 I FLOAT DECIMAL(13),
              4 J FIXED BINARY(31,0),
            3 K CHARACTER(2),
            3 L FIXED BINARY(20,0),
          2 M,
            3 N,
              4 P FIXED BINARY(5),
              4 Q CHARACTER(5),
              4 R FLOAT DECIMAL(2),
            3 S,
              4 T FLOAT DECIMAL(15),
              4 U BIT(3),
              4 V CHARACTER(1),
            3 W POINTER,
          2 X PICTURE '$9V99';
```

論理レベルが一番下の小構造はGであるから，まず最初にGのマップが作られる。次いでEのマップ作成が行なわれ，この後順にN，S，C，Mのマップ作成が行なわれる。最後に大構造Aのマップが作られる。各々の構造ごとに，マップ作成の手順を示す表とこの手順を図解するためのダイヤグラムを収録してある。
最後に，Aの構造マップが，Aの先頭を基準とする各々のメンバーのオフセットを示す表の形で示されている。

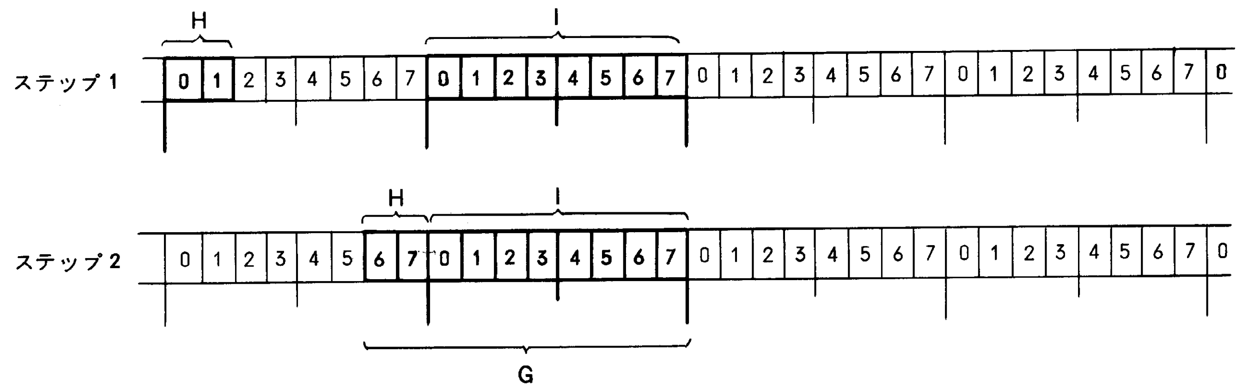
変数のタイプ	計算機内部では次のような形で貯えられる	所要記憶域の大きさ（バイト）	所要の境界合せ	注釈
BIT(n)	バイトの境界には関係なく，必要な数だけのビット	nビット	ビット	データはバイト0〜7の中の任意のビットから始まる
CHARACTER(n)	1文字につき1バイト	n	バイト	データは0〜7の任意のバイトから始まる
BIT(n)VARYING	2バイトの接頭語をもつBIT(n)と同じ	2バイト＋nビット		
CHARACTER(n) VARYING	2バイトの接頭語をもつCHARACTER(n)と同じ	2＋n		
PICTURE	PICTURE 文字1つにつき1バイト（ただし，VとKを除く）	VとKを除くPICTURE文字の数		
DECIMAL FIXED(p,q)	パック10進数形式（数字1つにつき1/2 バイト＋符号のための1/2 バイト）	CEIL((p+1)/2)		
BINARY FIXED(p,q) p<=15	ハーフワードの2進整数	2		
p>15	フルワードの2進整数	4		
BINARY FLOAT(p) p<22	短精度浮動小数点数			
DECIMAL FLOAT(p) p<7				
POINTER	−			
OFFSET	−			
FILE	−			
ENTRY	−	8		
LABEL	−			
BINARY FLOAT(p) 21<p<54	長精度浮動小数点数			
DECIMAL FLOAT(p) 6<p<17				

注：EVENTデータとAREA データにはUNALIGNED 属性を与えることができない。

図K.2 UNALIGNED データの境界合せの要件

	項目の名前	所要の境界合せ	長さ	ダブルワード境界からのオフセット		パディングの長さ	Gからのオフセット
				始り	終り		
ステップ1	H	バイト	2	0	1		
	I	ダブル	8	0	7		
ステップ2	*H	バイト	2	6	7		0
	I	ダブル	8	0	7	0	2
	G	ダブル	10	6	7		

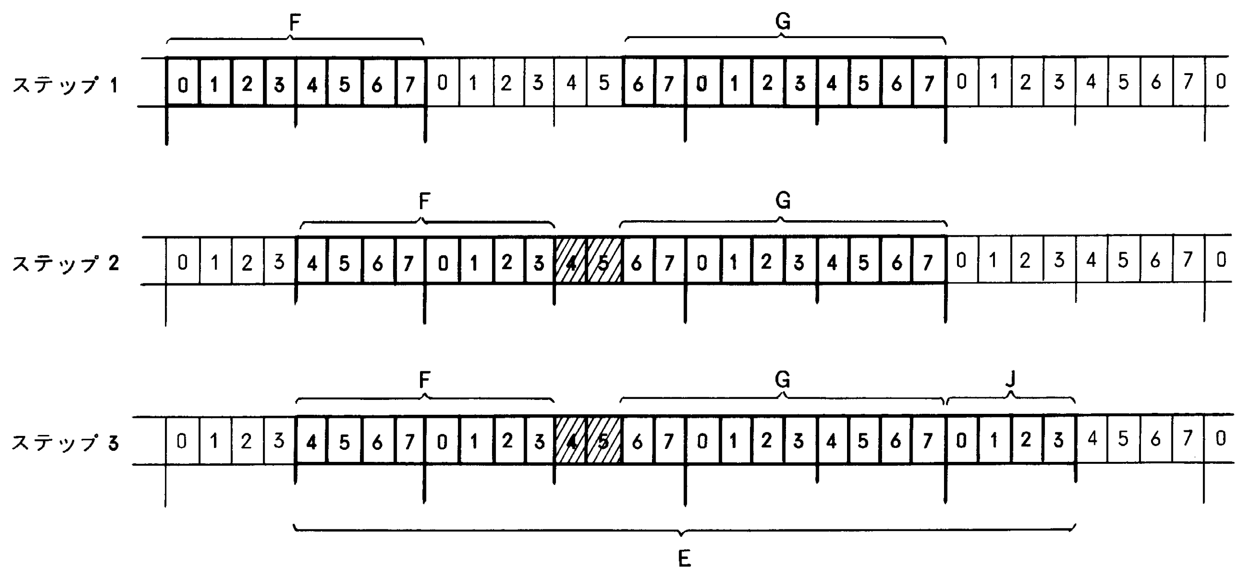
*最初の項目は右にずらされる



図K.3　小構造Gのマップ作成

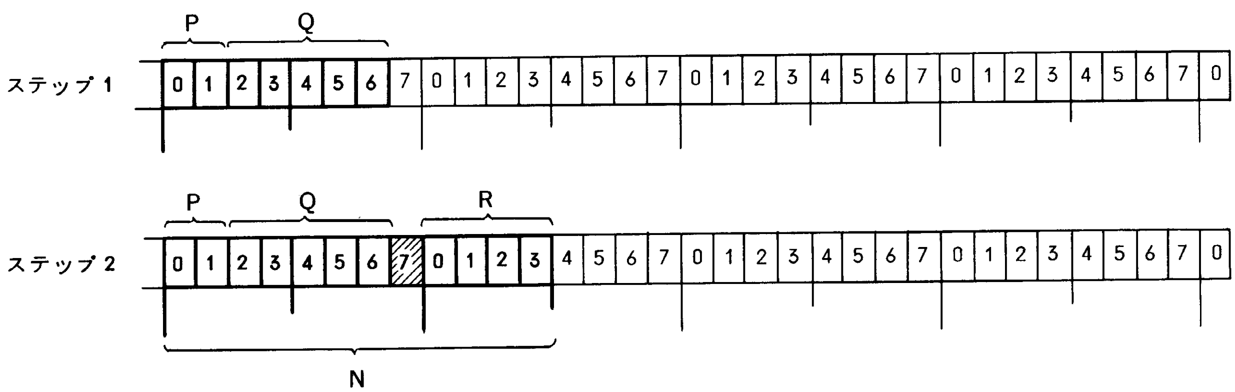
	項目の名前	所要の境界合せ	長さ	ダブルワード境界からのオフセット 始り	終り	パディングの長さ	Eからのオフセット
ステップ1	F	ワード	8	0	7		
	G	ダブル	10	6	7		
ステップ2	*F	ワード	8	4	3		0
	G	ダブル	10	6	7	2	10
ステップ3	F から G	ダブル	20	4	7		
	J	ワード	4	0	3	0	20
	E	ダブル	24	4	3		

*最初の項目は右にずらされる



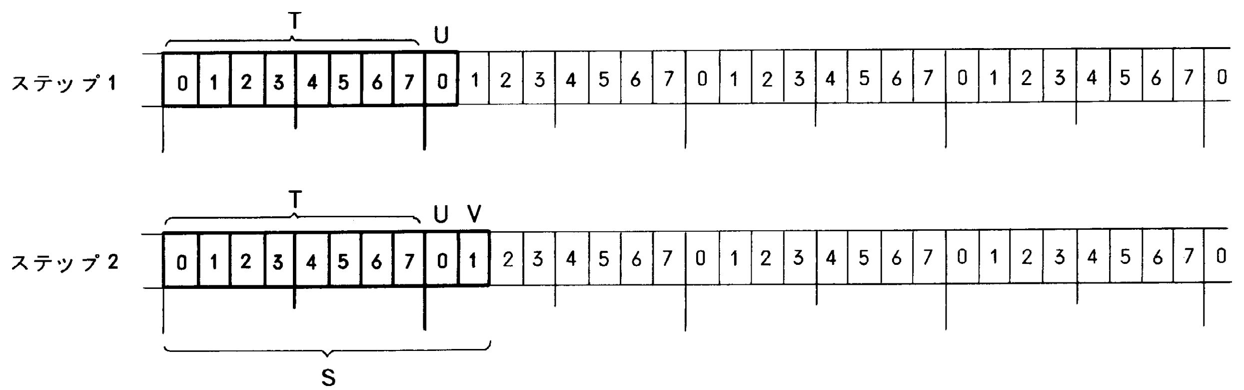
図K.4　小構造Eのマップ作成

	項目の名前	所要の境界合せ	長さ	ダブルワード境界からのオフセット		パディングの長さ	Nからのオフセット
				始り	終り		
ステップ1	P	ハーフワード	2	0	1		0
	Q	バイト	5	2	6		2
ステップ2	PからQ	ハーフワード	7	0	6		
	R	ワード	4	0	3	1	8
	N	ワード	12	0	3		



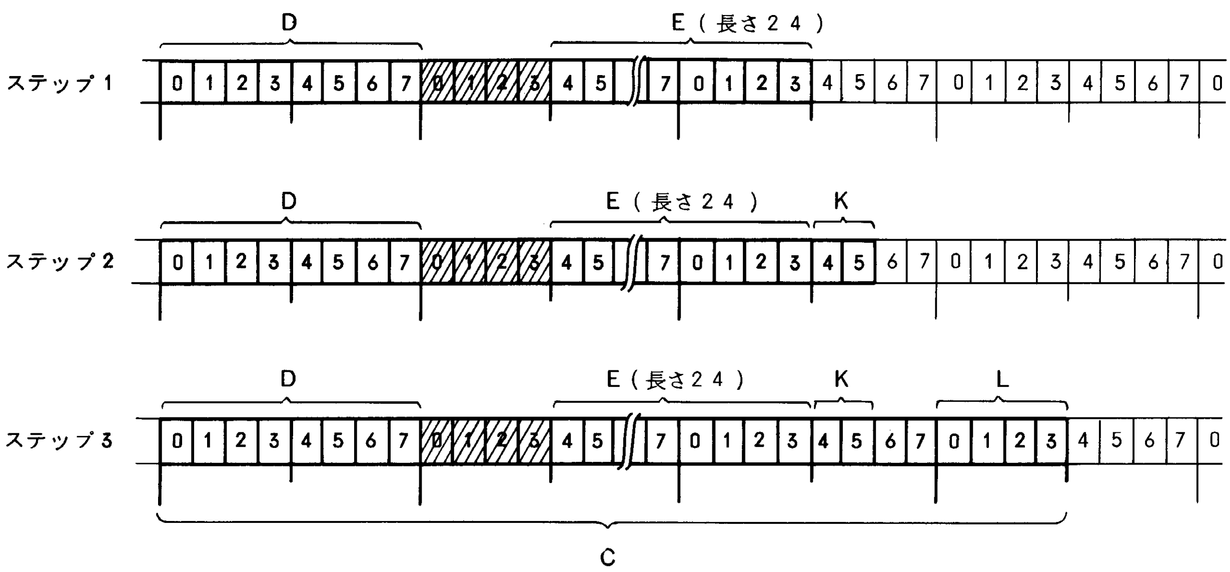
図K.5 小構造Nのマップ作成

	項目の名前	所要の境界合せ	長さ	ダブルワード境界からのオフセット 始り	終り	パディングの長さ	Sからのオフセット
ステップ1	T	ダブル	8	0	7		0
	U	バイト	1	0	0	0	8
ステップ2	TからU	ダブル	9	0	0		
	V	バイト	1	1	1	0	9
	S	ダブル	10	0	1		



図K.6　小構造Sのマップ作成

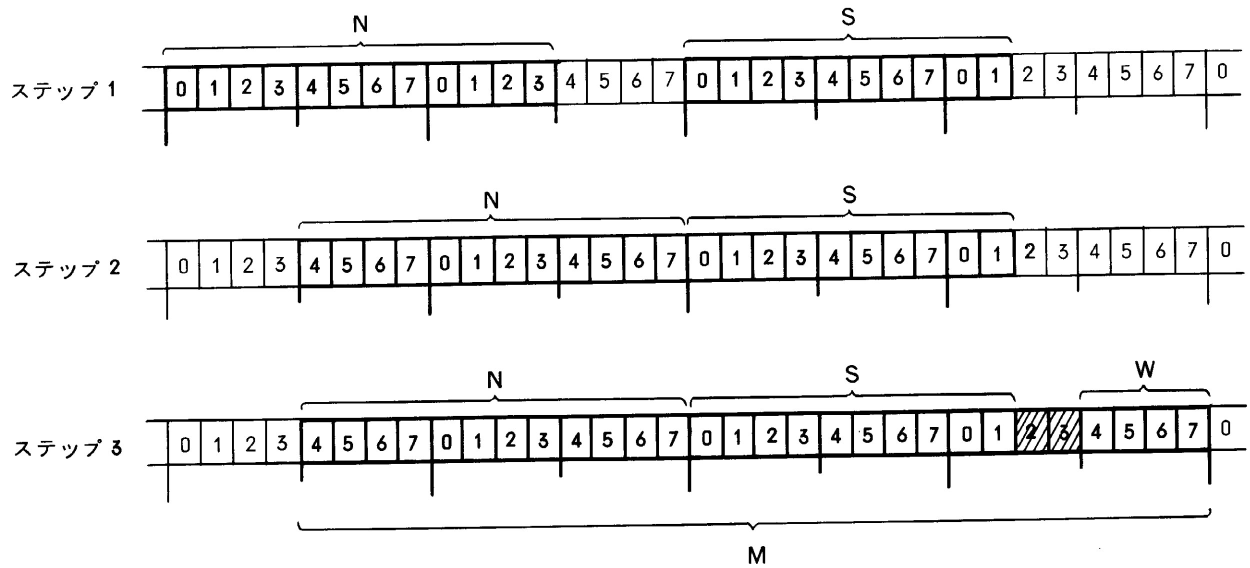
	項目の名前	所要の境界合せ	長さ	ダブルワード境界からのオフセット		パディングの長さ	Cからのオフセット
				始り	終り		
ステップ1	D	ダブル	8	0	7		0
	E	ダブル	24	4	3	4	12
ステップ2	DからE	ダブル	36	0	3		
	K	バイト	2	4	5	0	36
ステップ3	DからK	ダブル	38	0	5		
	L	ワード	4	0	3	2	40
	C	ダブル	44	0	3		



図K.7 小構造Cのマップ作成

	項目の名前	所要の境界合せ	長さ	ダブルワード境界からのオフセット		パディングの長さ	Mからのオフセット
				始り	終り		
ステップ1	N	ワード	12	0	3		
	S	ダブル	10	0	1		
ステップ2	*N	ワード	12	4	7		0
	S	ダブル	10	0	1	0	12
ステップ3	N から S	ダブル	22	4	1		
	W	ワード	4	4	7	2	24
	M	ダブル	28	4	7		

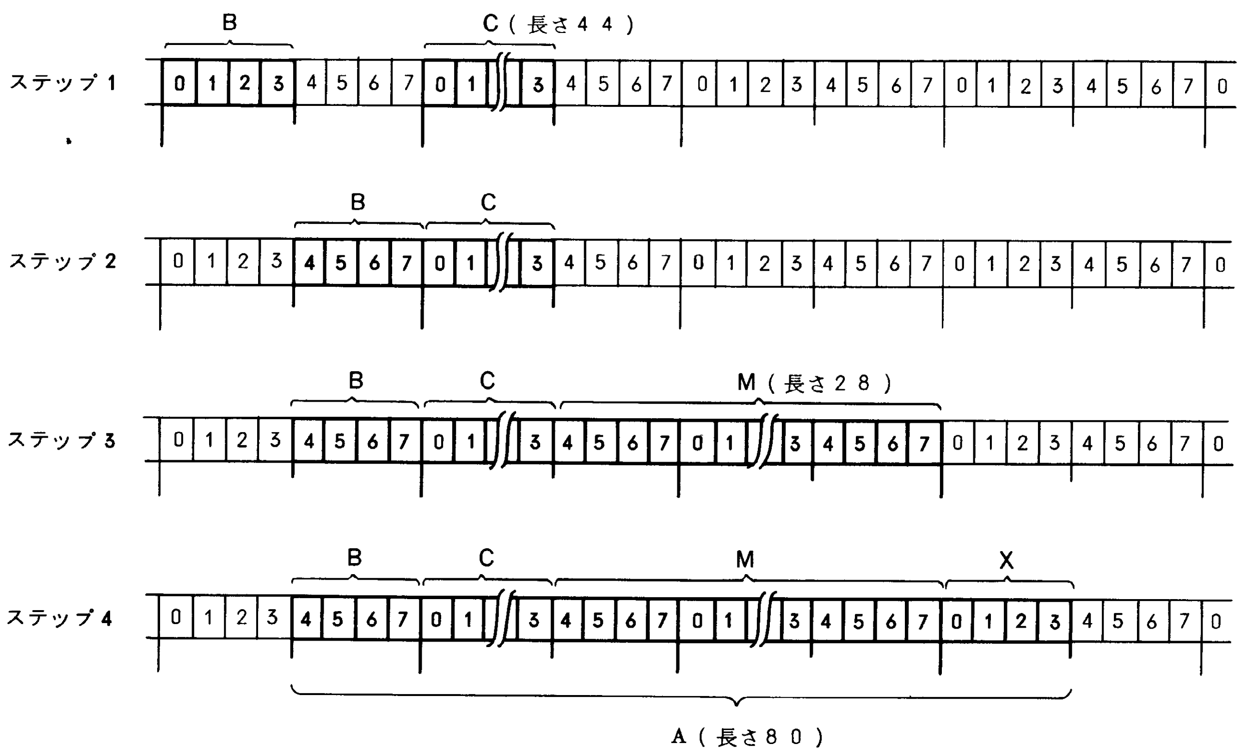
*最初の項目は右にずらされる



図K.8 小構造Mのマップ作成

	項目の名前	所要の境界合せ	長さ	ダブルワード境界からのオフセット 始り	終り	パディングの長さ	Aからのオフセット
ステップ1	B	ワード	4	0	3		
	C	ダブル	44	0	3		
ステップ2	*B	ワード	4	4	7		0
	C	ダブル	44	0	3	0	4
ステップ3	B から C	ダブル	48	4	3		
	M	ダブル	28	4	7	0	48
ステップ4	B から M	ダブル	76	4	7		
	X	バイト	4	0	3	0	76
	A	ダブル	80	4	3		

*最初の項目は左にずらされる



図K.9 大構造Aのマップ作成

A				Aから
B				0
C			Cから	4
D			0	4
パディング(4)			8	12
E		Eから	12	16
F		0	12	16
パディング(2)		8	20	24
G	Gから	10	22	26
H	0	10	22	26
I	2	12	24	28
J		20	32	36
K			36	40
パディング(2)			38	42
L			40	44
M			Mから	48
N		Nから	0	48
P		0	0	48
Q		2	2	50
パディング(1)		7	7	55
R		8	8	56
S		Sから	12	60
T		0	12	60
U		8	20	68
V		9	21	69
パディング(2)			22	70
W			24	72
X				76

図K.10　大構造Aの最終的なマップ作成におけるオフセット

レコードの境界合せ

位置指定方式の入出力を取り扱う場合には，ユーザーは緩衝域内でのレコードの境界合せに注意しなければならない。緩衝域の中では，あるブロックの中の最初のレコードの最初のデータ・バイトは通常，ダブルワード境界に揃えられる(図K.14 を参照のこと)。次のレコードは，緩衝域の中で次に使用可能なバイトから始まる。ユーザーの側では，このバイトの境界が，このレコードと結び付けられる基底付変数の所要境界と一致するよう配慮しなければならない。

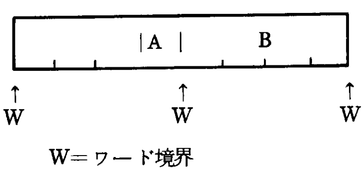
ブロック化レコードの場合は，レコード長(RECSIZE)が8の倍数であるならば，ブロック内の各々のレコードの最初のデータ・バイトは必ず，ダブルワード境界に揃えられる。ASCIIデータ・セットから読み取られたデータに関しては，ブロック接頭語の最初のバイトがダブルワード境界に揃えられる。最初のレコードの最初のバイトもダブルワード境界に揃えられるようにするには，接頭語の長さは8バイトの倍数から4を引いた値でなければならない(この4は，レコード長バイトの長さである)。

以下で説明する境界合せの問題の多くは，ALIGNED 属性をもつ基底付変数ないしALIGNED 属性をもつ基底付でない変数において見られるものである。これらの変数がUNALIGNED 属性をもっている場合には，緩衝域においてレコードの所定の境界を維持するのは前者にくらべてはるかに容易である。

次のような構造によって定義されている論理レコードを用いてVB形式レコードを作成するものとしよう。

```
1 S,
  2 A CHAR(1),
  2 B FIXED BINARY(31,0);
```

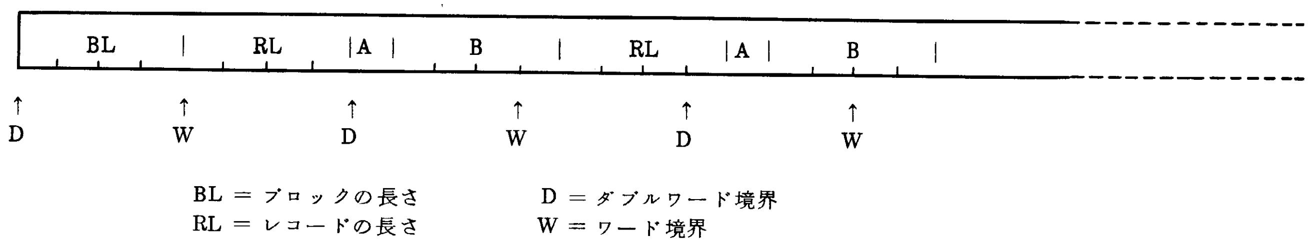
この構造のマップは図K.11 のようになる。



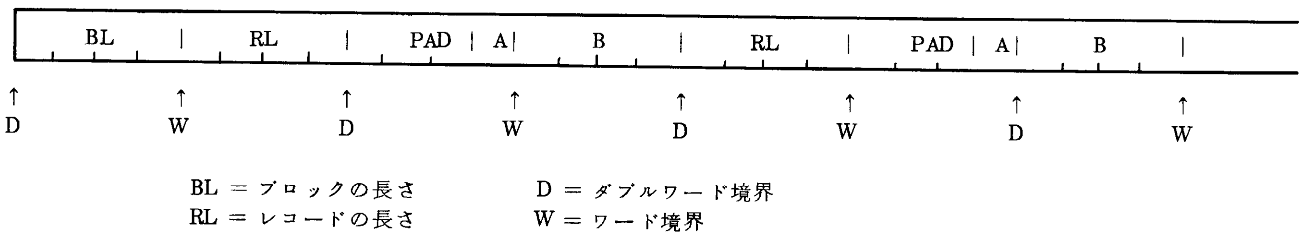
図K.11　構造Sの形式

一連のWRITE FROM(S)ステートメントを用いてブロックが作成されたとすると，このブロックの形式は図K.12 に示したようになる。この図から，緩衝域における境界合せとSの境界合せとが異なっていることがわかるであろう。

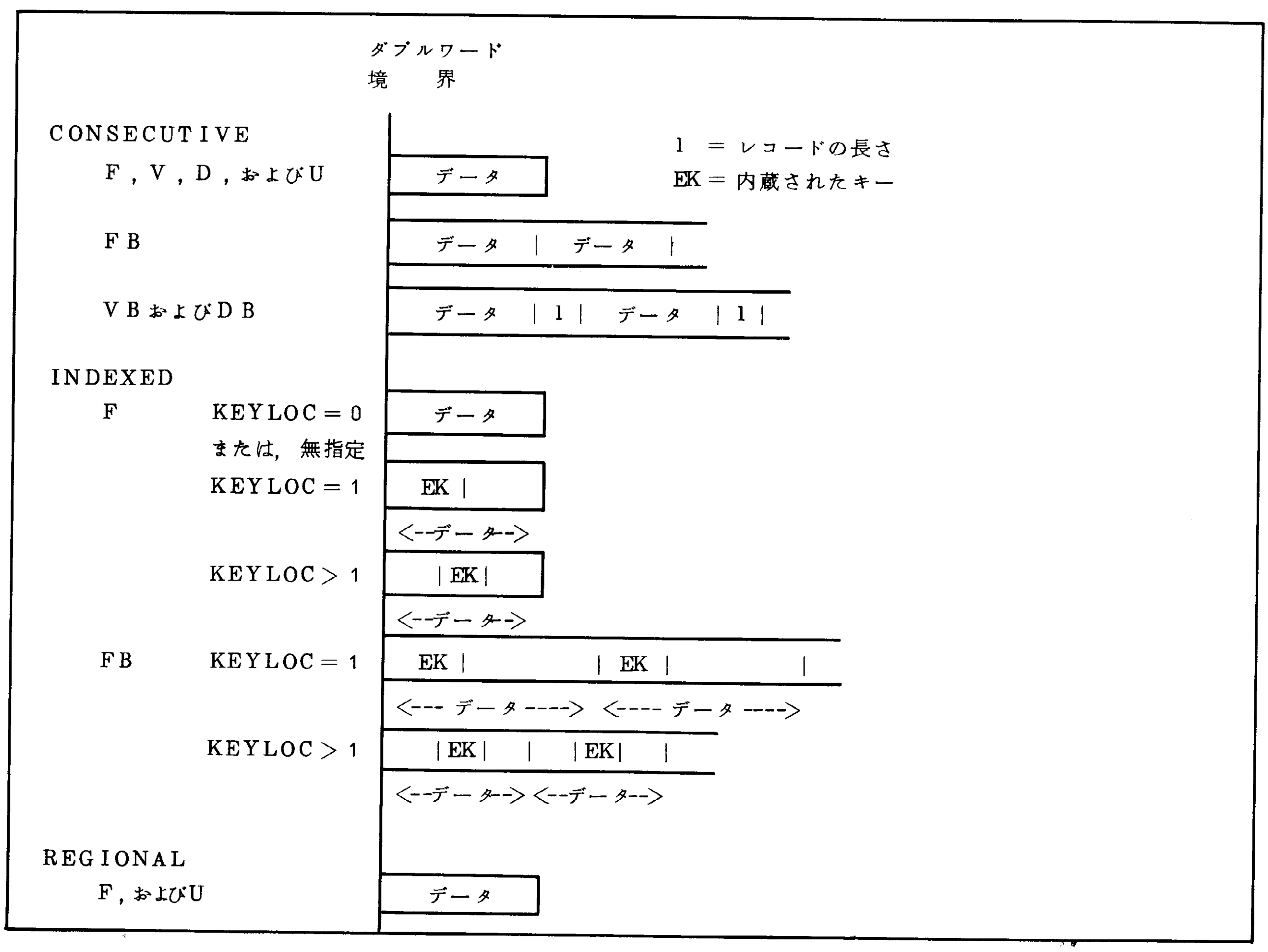
移動方式のREADステートメント(たとえば，READ INTO(S))を用いてファイルを読み取るのであれば問題はない。何故ならば，



図K.12　構造Sから作成されるブロック



図K.13　構造Sによって作成される，正しく境界合せされたブロック



図K.14　位置指定方式入出力における緩衝域内のデータの境界合せ
（形式別，データ・セット編成別）

情報は緩衝域から正しく境界合せされた記憶域へ移されるからである。

しかし，たとえば，次のように定義された構造があるときに，

```
1 SBASED BASED(P) LIKE S;
```

READ SET(P)ステートメントを使用した場合には，　ブロック内の最初のレコードとしてSBASED.B を参照すると，ダブルワード＋1バイトに揃えられたデータを参照したことになり，恐らく，指定割込みを引き起こすことになるであろう。

最初にファイルを作成するときに，次のステートメントを使用している場合にも，同じ問題が生ずるであろう。

```
LOCATE SBASED SET(P);
```

ここでも，ブロック内の最初のレコードに関して，Pはダブルワードを指すようセットされ，SBASED.B への参照は誤りになるであろう。

いずれの場合にも，Bがつねに正しい境界に揃えられるよう，この構造に次のようなパディングをしておけば，このような問題を回避することができる。

```
1 S,
  2 PAD CHAR(3),
  2 A CHAR(1),
  2 B FIXED BINARY(31,0);
```

こうすると，ブロック形式は図K.13 のようになり，Bはつねにワード境界に揃えられる。正しい境界合せを維持するために，構造の始りと終りにパディングが必要になる場合もある。

緩衝域内における種々のタイプのレコードの境界合せを図K.14に示す。データ・セットの編成およびレコード・タイプに関係なく，ブロック内の最初のデータ・バイトはつねにダブルワード境界に揃えられる。これ以降のレコードの緩衝域における位置は，レコード形式によって決まる。

索引ファイルを対象として順次方式の入力ないし更新をする場合は，あふれ域からレコードが取り出されることがある。この種のレコードには，冒頭に，10バイトの順序連係フィールドがついているので，READ SET ステートメントは，隠された緩衝域を使用して，各レコードの最初のバイトがダブルワード境界に揃えられるようにする。

調節可能なエクステントを含んでいる基底付変数（すなわち，REFERオプションをもつ基底付変数）と結びつけて位置指定方式入出力を使用する場合には，特殊な問題が生ずる。たとえば，次のような構造があるものとしよう。

```
1 S BASED(P),
  2 N,
  2 C CHAR (L REFER (N));
```

位置指定方式入出力を用いてこのタイプのV形式ブロック化レコードを作成するとすれば，レコードの境界合せは，Nがハーフワード境界に揃えられるようなものでなければならない。Lが2の倍数でない場合，現在のレコードの境界合せが正しければ，次のレコードの境界合せは正しくなくなる。しかし，次のような工夫をすれば境界合せを正しくすることができる。

```
LENGTH = L;
        /* SAVE DESIRED LENGTH L */
L = 2* CEIL(L/2);
        /* ROUND UP TO MULTIPLE OF 2 */
LOCATE S FILE(F);
N = LENGTH;
        /* SET REFER VARIABLE */
```

他の文脈でREFERオプションを使用している場合にも，この手法を応用することができる。

L節： 用語集

アクセス（access）： データを参照してからこれを取り出すに至るまでの操作。

値の計算（evaluation）： 式をある1つの値，または，値の配列，または，値の構造に移しかえること。

アドレス（address）： 記憶装置内において，データ項目を貯えることのできる特定の場所。

暗示的な解放（implicit opening）： OPENステートメント以外の入出力ステートメントの結果として，ファイルを開くこと。

暗黙宣言（implicit declaration）： 明示宣言も文脈宣言も行なわれていない識別名を，名前として定義させること。この識別名には一組の省略時属性が適用される。

入口式（entry expression）： 式の値を計算した結果が入口値であるような式。

入口定数（entry constant）： 入口名。

入口値（entry value）： 入口定数によって表わされる入口点。この値には，当該入口定数と結び付いている活用化の環境も含まれる。

入口点（entry point）： プロシージャーの呼出しをその地点から行なうことができるような，プロシージャー内の地点。（主入口点，副次入口点を参照のこと。）

入口変数（entry variable）： 入口値を表わすことのできる変数。この変数はENTRY，VARIABLE 両属性をもつものでなければならない。

入口名（entry name）： （VARIABLE属性が与えられていない場合）ENTRY属性をもつものとして明示的ないし文脈的に宣言されている識別名，または，ENTRY属性が暗示されている識別名。入口変数の値。

引数（argument）： プロシージャー参照の一部をなす引数リストの中の式。

因数化（factoring）： いくつかの名前を括弧で囲んで入れたリストに，1つ以上の属性または1つのレベル番号を適用すること。

引数リスト（argument list）： 入口名定数，入口名変数，総称，または組込み関数名の後に記入される，コンマで区切られた1つ以上の引数を括弧でくくったリスト。このリストは入口点のパラメーターに渡される。

インターリーブされた（非連続の）配列（interleaved array）： 連続していない記憶域を参照している名前をもつ配列。

英字（alphabetic character）： 英語のアルファベットA～Zとアルファベットの付属文字＃，$，および@の中の任意の文字（付属文字の表示は国により異なることがある）。

英数字（alphameric character）： 英字または数字。

エクステント（extent）：

1. 配列の次元の限界によって示される範囲，ストリングの長さによって示される範囲，またはエリアの大きさによって示される範囲。
2. エリア内の有効な割振り。

エピローグ（epilogue）： ブロックが終了するとき自動的に行なわれる操作。

エリア（area）： エリア変数によって識別される，連続した主記憶域内の宣言されている部分であり，割振りのときに，基底付変数を割り振るために確保される。

エリア変数（area variable）： AREA属性をもつ変数。この変数の値はエリアに限られている。

遠隔形式項目（remote format item）： 形式リストの中で，他のFORMATステートメントのラベルといっしょに指定される，Rという文字。

演算子（operator）： 行うべき演算を指定する記号。算術演算子，ビット・ストリング演算子，比較演算子，連結を参照のこと。

演算式（operational expression）： 1つ以上の演算子を含んでいる式。

オプション（option）： ステートメントの中で指定して，そのステートメントの実行方法ないし解釈に何らかの影響を与えるための指定。

オフセット変数（offset variable）： OFFSET 属性をもつロケーター変数。この変数の値は，エリアの始りを基準にして，記憶域内の場所を，識別するものである。

オペランド（operand）： その値に演算子が適用される式。

オン・ユニット（on-unit）： これが含まれているONステートメントで指名されているON条件が検出されたときにとられる処置。ただし，SYSTEMとSNAPを除く。

回帰的プロシージャー（recursive procedure）： まだ活用状態にある間に再度活用化することのできるプロシージャー。

解禁状態（enabled）： あるON条件が引き起された結果プログラム割込みが引き起こされるような状態をいう。

介在添字（interleaved subscripts）： 添字付の修飾された名前の場合に使用される添字表記法であり，それに必要なすべての添字がその構成要素名の直ぐ後に記入されていないもの。

開始ブロック（begin block）： BEGINステートメントに始まりENDステートメントで終る一連のステートメントの集りで，プログラムの一部をなすと共に，名前の有効範囲を限定し，制御の正規の逐次的な流れによって（GO TO ステートメントに起因するブランチも含む）活用化されるもの。

外部プロシージャー（external procedure）： 他のプロシージャーに含まれていないプロシージャー。

外部名（external name）： 有効範囲が，必ずしも，1つのブロックならびにこれに含まれているブロックに限定されていない（EXTERNAL属性をもつ）名前。

開放（ファイルの）（opening（of a file））： ファイルとデータ・セットとを結び付け，このファイル名に与えるべき属性のすべてを確定すること。

返される値（returned value）： 関数プロシージャーによって呼出し点に返される値。

下限（lower bound）： 配列のある次元の下方の限界。

仮想上の小数点ピクチャー文字（virtual point picture character）： ピクチャー指定において10進小数点の仮想上の位置を示すのに使用されるピクチャー指定用文字V。

活用化（ブロックの）（activation（of a block））：

1. ブロックを活用化するための操作。

2. ブロックの実行。

活用化（プリプロセッサー変数または入口名の）（activation（of a preprocessor variable or entry name））： 変数の値の置換えまたは入口名の返された値の置換えが可能な状態にすること。最初の活用化は，%DECLAREステートメントに名前が出て来た結果行なわれたものでなければならない。変数または入口名の状態が，%DEACTIVATEステートメントによって，活用状態から非活用状態に切り替えられた後，%ACTIVATEステートメントによってこれを再び活用化することができる。

活用化する（ブロックを）（activate（a block））： ブロックの実行を開始すること。プロシージャー・ブロックは，そのいずれかの入口点を用いて呼び出されたときに活用化される。開始ブロックは，正規の制御の流れ（ブランチを含む）の中にその開始ブロックが出て来たときに活用化される。

活用状態（active）：

1. 活用化の後，終了に至るまでのブロックの状態。

2. プリプロセッサー変数またはプリプロセッサー入口名の値によって原始プログラム・テキストの中のこれに対応する識別名を置き換えることができるときの，この変数または入口名の状態をいう。

3. 事象変数が非同期操作と結び付けられているときの，この変数の状態をいう。事象変数は，この事象変数を指定しているWAITステートメントが実行されるまでは，活用状態にとどまり，この間，この変数を他の操作と結び付けることはできない。

環境（活用化の）（environment（of an activation））： ブロックの呼出しに関連する情報であり，呼び出されたブロックの中で，このブロックの外部で宣言されているデータへの参照の解釈に使用されるもの。この情報には，自動変数の諸世代，被定義変数のエクステント，パラメーターの諸世代が含まれる。

環境（ラベル定数の）（environment（of a label constant））： ステートメント・ラベル定数への参照が適用される，ブロックのある特定の呼出しの識別。ステートメント・ラベル定数が引数として渡されるとき，またはステートメント・ラベル定数がステートメント・ラベル変数に割り当てられるときに，この情報は確定し，ステートメント・ラベル定数といっしょに渡されるかまたは割り当てられる。

緩衝域（buffer）： 入出力操作において使用される中間記憶域であり，入力のときにはレコードがそこに読み込まれ，出力のときにはレコードがそこに書き出される。

関数（function）： 関数プロシージャー（プログラマーが指定した関数プロシージャーまたは組込み関数プロシージャー）。関数参照で指定されているいずれか1つの入口名によって呼び出され，参照点に値を返すプロシージャー。

関数参照（function reference）： 式の中に入口名または組込み関数名（ないし，入口変数）を記入すること。

完全に修飾された名前（fully-qualified name）： 完全な形の修飾された名前。すなわち，この名前で指そうとしている構造のメンバーの名前と，さらに構造の組立ての上からみてそのメンバーより上位にあるすべての名前を順に含んでいる，修飾された名前。

キー（key）： 直接アクセス・データ・セット内のあるレコードを識別するためのデータ。原始キーおよび記録されたキーを参照のこと。

キーワード（keyword）： PL／Iの一部をなしており，適切な文脈で使用されているとコンパイラーに特定の意味を伝える識別名。

擬似変数（pseudovariable）： 目標変数を指定するのに使用することができる組込み関数名。

記述子（descriptor）： パラメーター記述子を参照のこと。

基数（base）： 算術値を表現するための記数法。

基底項目（base item）： 被定義変数をその上に重ねて定義される自動変数，被制御変数，静的変数，またはパラメーター。この名前は，修飾されていても，添字が付いていてもよいし，また修飾と添字の両方がついてもよい。

基底付記憶域の割振り（based storage allocation）： 基底付変数のための記憶域を割り振ること。

基底付変数（based variable）： ロケーター変数によって識別される世代をもつ変数。基底付変数は，どの記憶域クラスにある変数の値を参照するときにも使用することができる。ALLOCATEステートメントおよびFREEステートメントを使用して，基底付変数を明示的に割り振ったり，明示的に解放したりすることができる。

基本項目名（elementary name）： 基本要素を参照のこと。

基本要素（base element）： 構造のメンバーのうち，小構造でないものを表わすための名前。

境界合せ（alignment）： ある機械依存の境界を基準にしてデータ項目を貯えること。

切捨て（truncation）： 目標変数のストリングの長さまたは精度を超過したとき，データの項目の一端から１つ以上の数字，文字，またはビットを取り除くこと。

記録されたキー（recorded key）： 対応するデータ・レコードを識別するために，直接アクセス・ボリュームに記録されているキー。

禁止状態（disabled）： ある特定のON条件が引き起されても，割込みが起こらないような状態を言う。

区切り符号（delimiter）： すべての演算子，注釈，および以下にあげる符号。パーセント，括弧，コンマ，ピリオド，セミコロン，コロン，割当て記号，ブランク。これらの符号は，識別名，定数，ピクチャー指定，iSUB，およびキーワードの境界を定義する。

組込み関数（built-in function）： PL／Iによって提供される関数。

組立て（structuring）： ある構造のメンバーの数，メンバーが出てくる順序，メンバーの属性，メンバーの論理レベル，などの点から見たその構造の骨組み（ただし，メンバー名や宣言されているレベル番号は必ずしもこの要件とはならない）。

繰返し因数（repetition factor）： 次の事項を指定するための無符号の１０進整数定数を括弧で囲んだもの。

1. あるストリング定数があるストリング構成の繰返しで構成されている場合，その繰返し回数。
2. ピクチャー指定におけるピクチャー指定用文字の繰返し回数。

繰返しの指定（repetitive specification）： １つ以上のデータ項目を一定条件のもとで繰り返し転送するよう指定するためのデータ・リスト内の要素で，通常，配列の場合に使用される。

グループ（group）： DOグループ。オン・ユニットとしてDOグループを使用することだけはできないが，これ以外の場所であれば，単独のステートメントを置くことができるところであれば，どこでも使用することができる。

形式項目（format item）： 編集指示転送において，流れ内のデータ項目の表現について記述するのに使用される指定（データ形式項目），または流れ内のデータ項目の位置設定を指定するのに使用される指定（制御形式項目）。

形式項目の制御（control of format item）： 編集指示転送において，流れの中のデータ項目または印刷されるページの上のデータ項目の位置を指定するのに使用される指定。

形式リスト（format list）： 編集指示データ指定に必要ないくつかの形式項目を括弧で囲んだリスト。

限界（bounds）： 配列の次元の上限と下限。

言語文字セット（language character set）： プログラム要素を原始言語で表わすために定義されている文字セット（ここでは，文字ストリング定数と注釈はプログラム要素とはみなさない）。

現在の世代（current generation）： 自動変数または被制御変数の名前を参照したときに，現時点で，アクセスされるこの変数の世代。

原始キー（ source key ）： 直接アクセス・データ・セットの中のある特定のレコードを識別するために，レコード式転送ステートメントの中で参照されるキー。

原始プログラム（ source program ）： コンパイラーへの入力となるプログラム。原始プログラムにプリプロセッサー・ステートメントが含まれていることもある。

原変数（ source variable ）： 割り当てるべき値をもつ変数または他の何らかの操作に関与する値をもつ変数。

構造（ structure ）： 階層に分けられた名前の集りで，データ項目の集合体を参照するためのものである。この個々のデータ項目の属性は異なっていてもかまわない。

構造式（ structure expression ）： 式の値を計算した結果が値の構造になるような式。

構造の配列（ array of structures ）： 構造名に次元属性を与えることによって指定される同一の構造をいくつか集めて並べたもの。

構造のメンバー（ structure member ）： 構造の中の任意の小構造または要素名。

コード化算術データ（ coded arithmetic data ）： 変換をしなくても算術計算を行なうことのできる形で貯えられている算術データ。

固定小数点定数（ fixed-point constant ）： 算術定数を参照のこと。

コンパイル時間（ compile time ）： 一般に，プログラムが目的モジュールに翻訳されている時間。PL／Iの場合は，必要に応じて原始プログラムを変更し（プリプロセッサーで処理し），次いでこれを目的プログラムに翻訳する時間。

コンパイル時のステートメント（ compile-time statements ）： プリプロセッサー・ステートメントを参照のこと。

サブフィールド（ピクチャー指定の）（ subfield(of picture specification) ）： ピクチャー指定用フィールドの中のピクチャー指定用文字Vの前後の各部分。

サブルーチン（ subroutine ）： CALLステートメントまたはCALLオプションによって呼び出されるプロシージャー。サブルーチンは，呼出しブロックに値を返すことはできないが，変数の値を変更することはできる。

算術演算子（ arithmetic operators ）： 接頭演算子＋または－，または，次にあげる挿入演算子のいずれか1つ。＋　－　*　／　**。

算術定数（ arithmetic constant ）： 固定小数点定数または浮動小数点定数。殆んどの算術定数は符号をつけることができるが，符号は定数の一部ではない。

算術データ（ arithmetic data ）： 基数，スケール，モード，および精度をもつデータ。これには，コード化算術データおよびピクチャー数字データが含まれる。

算術ピクチャー・データ（ arithmetic picture data ）： 10進ピクチャー・データまたは2進ピクチャー・データ。

算術変換（ arithmetic conversion ）： 値の表現形式を，ある算術表現から他の算術表現にかえること。

参照（ reference ）： 明示宣言となるような文脈以外のところに名前が現われること。

式（ expression ）： プログラムの中で使用される，1つの値，値の配列，または値の構造を表わす式。単独の定数あるいは参照，または，定数および／または参照を演算子で結び合わせたもの。

識別名（ identifier ）： 英数字および，場合によっては，分割文字からなるストリングで，注釈あるいは定数の中に含まれておらず，前と後に分離符号があるもの。その頭文字は英字でなければならない。

次元数（ dimensionality ）： 配列の宣言の中にある限界の指定の数。

自己定義データ（ self-defining data ）： データ自体の属性について記述している情報（たとえば，調節可能な限界や長さの値）が含まれているデータ項目ないしデータ項目の集合体。

事象（ event ）： その状況と完了がそれと結び付いている事象変数から決められるようなプログラム内の活動。

事象変数（ event variable ）： ある事象と結び付けることができる，EVENT属性をもつ変数。この変数の値は，その活動が完了したかどうかということ，および完了の状況を表わす。

指数文字（ exponent characters ）： 次のピクチャー指定用文字。

1. K，E。浮動小数点ピクチャー指定において，指数フィールドの始りを示すのに使用される。

2. F。整数定数と共に指定されるスケール因数文字。この整数定数は，（正の場合には）10進小数点が仮想上の10進小数点から何桁右にあるかを表わし（負の場合には）何桁左にあるかを表わす。

システムの標準処置（standard system action）： あるON条件に対するオン・ユニットが指定されていないときにとるよう，この言語で指定されている処置。

自動記憶域の割振り（automatic storage allocation）：自動変数に対して記憶域を割り振ること。

自動変数（automatic variable）： ブロックが活用化されるときに自動的に割り振られ， このブロックの終了と共に自動的に解放される変数。

主入口点（primary entry point）： PROCEDUREステートメントのラベル・リストにあるいずれかの名前によって識別される入口点。

集合体（aggregate）： データ集合体を参照のこと。

集合体式（aggregate expressions）： 配列式または構造式。

修飾された名前（qualified name）： 構造内の構成要素を識別するのに使用される名前であり，構造のメンバーの名前を，構造内での順位にしたがって，ピリオドでつなげたもの。メンバー名は添字付の名前でもよい。ロケーター修飾の項も参照のこと。

終了（ブロックの）（termination（of a block））： ブロックの実行を停止すると共に，このブロックを活用化したブロックにRETURNないしENDステートメントによって制御を返すか，あるいはこのブロックを活用化したブロックもしくは活用状態にある他のブロックにGO TOステートメントによって制御を渡すこと。

条件（condition）： ON条件を参照のこと。

上限（upper bound）： 配列の次元の上の限界。

条件接頭語（condition prefix）： ステートメントの接頭語として使用される，1つ以上のPL／I条件名を括弧で囲んだリスト。これは，指定されているON条件を解禁状態に置くかどうか指定するためのものである。

条件名（condition name）： プログラムを実行している間に起こる可能性のあるON条件を表わすPL／Iキーワード（または，CONDITIONの後にプログラマー定義の名前を括弧で囲んで記入したもの）。

条件リスト（condition list）： 1つ以上の条件接頭語からなるリスト。

小構造（minor structure）： 他の構造に含まれている構造。小構造の名前は，1より大きいレベル番号によって宣言される。

定数（constant）： 名前をもたず，また値が変わることもない算術データ項目またはストリング・データ項目。添字の付かないラベル接頭語，または，ファイル名，または，入口名。

商用文字（commercial character）： 以下のピクチャー指定文字。

1. CR （貸方）。
2. DB （借方）。
3. T.I.R.（上重ね穿孔符号文字）。これらは，データ項目の中の対応する位置には上重ね穿孔符号をつけた形で（つかないこともある）数字が入っていること，および，この上重ね穿孔符号もこのデータ項目の文字ストリング値に加えられることを指示するためのものである。

省略時属性（省略時オプション）（default）： 択一属性またはオプションが指定されていないときに，指定されるか，またはDEFAULTステートメントによってとるよう指定されている択一属性またはオプション。

処置の指定（action specification）： ONステートメントにおけるオン・ユニットないしSYSTEMというキーワード。いずれも，指定されたON条件が引き起されて割込みが起ったときにとるべき処置を指定するためのものである。処置の指定にSNAPというキーワードを含めてよい。

知られている（known）： （名前に関して用いられる） 宣言されている意味をもつものとして認定されていること。名前は，その有効範囲全体にわたって知られている。

数字（digit）： 文字0～9のいずれか1つ。

数字データ（numeric character data）： 10進ピクチャー・データを参照のこと。

スカラー項目（scalar item）： データの1項目。要素。

スカラー変数（scalar variable）： 1つのデータ項目しか表わすことのできない変数。要素変数。

スケール（scale）： 数学上の記数法。算術値の固定小数点の進法または浮動小数点の進法。

スケール因数(scale factor): 固定小数点数の小数部分の桁数の指定。

ステートメント(statement): PL/Iプログラムの基本要素であり，プログラムのある部分の境を示したり，プログラムの中で使用される名前について記述したり，とるべき処置を指定したりするのに使用される。ステートメントは，条件リスト，ラベル・リスト，ステートメント識別名，ステートメント本体(これはセミコロンで終る)などから構成することができる。

ステートメント識別名(statement identifier): ステートメントの目的を示すためのPL/I言語のキーワード。

ステートメント本体(statement body): ステートメントの内，ステートメント識別名(識別名がついていれば)の直ぐ後からセミコロンに至る部分。これにはステートメントのオプションも含まれる。

ステートメント・ラベル式(statement-label expression): ラベル式を参照のこと。

ステートメント・ラベル定数(statement-label constant): ラベル定数を参照のこと。

ステートメント・ラベル変数(statement-label variable): ラベル変数を参照のこと。

ストリング(string): 1つのデータ項目として取り扱われる，連続している文字またはビットの連なり。

ストリング変数(string variable): BIT属性またはCHARACTER属性をもつものとして宣言されている変数で，その値がビット・ストリングまたは文字ストリングをとることができるもの。

制御の流れ(flow of control): 実行順序。

制御変数(control variable): あるグループの実行の反復を制御するのに使用される変数。反復DOグループを参照のこと。

静的記憶域の割振り(static storage allocation): 静的変数のための記憶域の割振り。

静的変数(static variable): 当該プログラムの実行が開始される前に割り振られ，当該プログラムが実行されている間中割り振られたままになっている変数。

精度(precision): 算術変数の値域を総桁数(および，固定小数点変数に関しては，10進小数点ないし2進小数点の右側にあると想定される桁数)で表わしたもの。

世代(変数の)(generation (of a variable)): 静的変数の割振り，被制御変数または自動変数のある特定の割振り，基底付変数のある特定のロケーター修飾語が指す記憶域，または，被定義変数ないしパラメーターが指す記憶域。

接頭演算子(prefix operator): あるオペランドの直前に置かれ，このオペランドだけに適用される演算子。+(正)，-(負)，¬(否定)。

接頭語(prefix): ステートメントの冒頭にコロンでつながれている，ラベル，または，1つ以上の条件名を括弧で囲んだリスト。

ゼロ消去文字(zero suppression character): 対応する数字桁のゼロを消去するのに使用されるピクチャー指定用文字Z，Y，*。

宣言(declaration):

1. 識別名を名前として確定させると共に，この名前にそのすべてまたは一部の属性を与えること。
2. ある特定の名前に属性を与えるための1つの手段。

先行ゼロ(leading zeros): 算術整数の値の中にある，意味をもたないゼロ。ある数の最初の有効整数桁の左にあるすべてのゼロ。

総称名(generic name): 関係のあるいくつかの入口名からなる集団の名前。この名前への参照は，呼出し点の引数リストの中にある引数の属性と合致するパラメーター記述子をもつ特定の入口名と置き換えられる。

挿入演算子(infix operator): 2つのオペランドの間にある演算子。

挿入ピクチャー文字(insertion picture character): 対応するデータを文字ストリングに割り当てるときに所定の位置に挿入されるピクチャー指定文字。入力のためのP形式項目で使用されている場合は，この文字はチェック用ピクチャー文字の役割を果す。

添字(subscript): 配列のある次元の中のある位置を指定するための要素式。添字として星印を指定することもできる。この場合，この星印はこの次元の全範囲を指定するものである。

添字リスト(subscript list): 配列の各次元から1つずつとられた1つ以上の添字を括弧で囲んだリストであり，全体として，この配列の1つの要素または断面を一義的に識別するもの。

属性(attribute):

1. ある名前が表わす項目の特性を記述するためにその名前と結び付けられる記述上の性質。
2. ある式の値を計算して得られる結果の特性を記述するのに使用される記述上の性質。

第1プロシージャー(initial procedure): OPTIONS(MAIN)属性をもつPROCEDUREステートメントをもっている外部プロシージャー。どのPL/Iプログラムにも第1プロシージャーがなければならない。第1プロシージャーは，プログラムの実行にあたっての最初のステップとして自動的に呼び出される。

大構造(major structure): レベル番号1によって宣言されている名前をもつ構造。

択一属性(alternative attribute): 2つ以上の属性からなるグループの中から選び出すことのできる1つの属性。この属性を指定しないと，省略時属性がとられる。

単純パラメーター(simple parameter): 記憶域クラス属性が指定されていないパラメーター。このパラメーターはどの記憶域クラスの引数でも表わすことができるが被制御引数の現在の世代しか表わすことができない。

注釈(comment): 文書化のために使用されるゼロまたは1つ以上の文字からなるストリングで，/*で始まり*/で終り，プログラム内の区切りとなるもの。

調節可能なエクステント(adjustable extent): 関連する変数の世代ごとに異なる値をとることのできる(配列の)限界，(ストリングの)長さ，または(エリアの)大きさ。調節可能な限界，長さ，および大きさは，式または星印(または，基底付変数に関しては，REFERオプション)を使用して指定する。各世代ごとに，個々に，これらの値が計算される。静的変数に対しては，調節可能なエクステントを使用することはできない。

データ(data): 処理に適した形で情報ないし値を表現したもの。

データ形式項目(data format item): 編集指示転送において，流れの中のデータ項目の表現について記述するのに使用される指定。

データ項目(data item): 1単位のデータ。要素と同義。

データ指示転送(data-directed transmission): ストリーム方式転送の1つのタイプ。データは，各々コンマで区切られた1つ以上の項目からなり，セミコロンで終っている1つのグループとして転送される。各々の項目は次の形をしている。

名前 = 定数

名前は，修飾されていても添字付であってもよいし，あるいはその両方でもよい。

データ指定(data specification): ストリーム式データ転送ステートメントの中の転送のモード(DATA，LIST，またはEDIT)を指定する部分であり，1つ以上のデータ・リスト，および，編集指示転送の場合には1つ以上の形式リスト，を含む。

データ集合体(data aggregate): 全体としても，個々にも参照することのできる2つ以上のデータ項目の論理的な集り。配列または構造。

データ・セット(data set): 当該プログラムの外部にあり，当該プログラムにおいてファイル名を参照することによってアクセスすることのできるデータの集り。

データ転送(data transmission): データ・セットからプログラムへあるいはプログラムからデータ・セットへデータを移すこと。

データの流れ(data stream): 文字形式のデータ要素の連続した流れとして，ストリーム式転送によってデータ・セットとの間で転送されているデータ。

データ文字セット(data character set): 使用しているコンピューターによって認定される表現をもつすべての文字。

データ・リスト(data list): 入力の場合はデータ項目を割り当てるべき記憶場所を表わし，出力の場合は取得すべき値を表わすコンマで区切られたいくつかの式または繰返し指定を括弧で囲んだリスト――ストリーム方式入出力指定で使用される。

同期的(synchronous): あるプログラムを逐次的に実行するのに制御の流れを1つだけ使用すること。

内部テキスト(internal text): あるブロックに含まれているすべてのテキストから他のブロックに含まれているテキストを除いた残りのテキスト。したがって，内部ブロックのテキストは(その入口名を除けば)，このブロックを含んでいるブロックの内部にあるわけではない。

内部プロシージャー(internal procedure): あるブロックの中に含まれているプロシージャー。

内部ブロック(internal block): 他のブロックの中に含まれているブロック。

内部名(internal name): その名前の宣言が行なわれているブロックの外では知られていない名前。

流れ(stream): データの流れを参照のこと。

名前(name): キーワードにはならない文脈にある識別名。

ナル・ストリング(null string): 長さがゼロのストリング・データ項目。

ナル・ロケーター値(null locator value): 内部記憶装置内のどの記憶場所も識別することのできない特殊なロケーター値。この値は，ロケーター変数が現在データのどの世代も識別していないことを積極的に示すためのものである。

入出力(input/output): 補助記憶装置と主記憶装置の間でのデータの移動。

ネスティング(nesting):

1. あるブロックを他のブロックの中に置くこと。

2. あるグループを他のグループの中に置くこと。

3. IFステートメントをTHEN文節またはELSE文節の中に置くこと。

4. 関数参照を関数参照の引数とすること。

5. FORMATステートメントの形式リストの中に遠隔形式項目を置くこと。

6. あるパラメーター記述子のリストを他のパラメーター記述子のリストの中に置くこと。

7. 1つ以上の共通属性を因数化して括弧で囲んだ名前のリストの中で，ある属性を指定すること。

配列(array): すべて同じ属性をもついくつかのデータ要素を集めて並べ，この集り全体にある名前を与えたもの。配列は，次元属性によって指定される次元をもち，その個々の要素は添字によって参照される。配列は，また，同じ構造をいくつか集めて並べたものであってもよい。

配列式(array expression): 式を評価した結果が配列値になる式。

配列の構造(structure of arrays): 次元属性をもつものとして個々のメンバー名を宣言することにより指定されている配列を含んでいる構造。

配列の断面(cross section of an array): 配列の少なくとも1つの次元(しかし，すべての次元ではない)の範囲によって表わされる諸要素。配列参照において添字のかわりに星印を指定すると，その次元の範囲全体を参照することになる。

パック10進数(packed decimal): システム/360における固定小数点10進データ項目の内部表現。

パディング(padding): あるストリングを所要の長さに拡張するために，そのストリングの右側に連結される1つ以上の文字ないしビット。文字ストリングの場合は，ブランクが補われ，ビット・ストリングの場合にはゼロが補われる。

パラメーター(parameter): プロシージャーの中で，このプロシージャーに渡された引数を参照するのに使用される名前。

パラメーター記述子(parameter descriptor): ENTRY属性の指定の中で，ある1つのパラメーターに対して指定されている一組の属性。

パラメーター記述子リスト(parameter descriptor list): ある1つのENTRY属性の指定の中にあるすべてのパラメーター記述子を内容とするリスト。

パラメーター・リスト(parameter list): PROCEDUREステートメントのキーワードPROCEDUREまたはENTRYステートメントのキーワードENTRYの後にある，1つ以上のパラメーターをカンマで区切り括弧で囲んだリスト。このリストは，呼出しのとき渡される引数リストに対応している。

範囲(省略時指定の)(range(of a default specification)): DEFAULTステートメントの省略時指定で指定されている属性が適用される一組の識別名，定数，および/または，パラメーター記述子。

反復因数(iteration factor): 次の事項を指定するための式。

1. INITIAL属性の指定では，配列内の，与えられた定数で初期値化すべき連続している要素の数。

2. 形式リストでは，与えられた形式項目ないし形式項目のリストを繰り返し続けて使用すべき回数。

反復DOグループ(iterative do-group): 制御変数および/またはWHILEオプションを指定しているDOステートメントをもつDOグループ。

比較演算子(comparison operators): 比較式の中で使用される挿入演算子。これは次のようなものがある。¬<(小さくない),<(小さい),<=(小さいか等しい),()(等しくない),¬=(等しい),>=(大きいか等しい),>(大きい),¬>(大きくない)。

非活用状態(deactivated): プリプロセッサー変数またはプリプロセッサー入口名が,その値によって原始プログラム・テキストの中のこれに対応する識別名を置き換えることができない状態をいう。

ピクチャー指定(picture specification): 10進ピクチャー・データおよび文字ストリング・ピクチャー・データの構成および特性を各文字ごとに記述したもの。

ピクチャー指定用の文字(picture specification character): ピクチャー指定で使用することができる文字。10進ピクチャー・データおよび文字ストリング・ピクチャー・データを参照のこと。

被制御記憶域の割振り(controlled storage allocation): 被制御変数のための記憶域の割振り。

被制御パラメーター(controlled parameter): DECLAREステートメントの中でCONTROLLED属性が指定されているパラメーター。このパラメーターと結び付けることのできる引数はCONTROLLED属性をもつものだけである。

被制御変数(controlled variable): ALLOCATEおよびFREEステートメントによって割振りと解放が行なわれる変数。この変数に関しては,その現在の世代にしかアクセスすることができない。

ビット(bit): 2進桁(0または1)。

ビット・ストリング(bit string): ゼロまたは1つ以上のビットからなるストリング。

ビット・ストリング演算子(bit-string operators): 論理演算子¬(not),&(and),および|(or)。

被定義項目(defined item): 基底項目と呼ばれる他の変数に割り振られている記憶域の一部またはすべてを表わすよう宣言されている変数。

非同期操作(asynchronous operation): 入出力操作とステートメントの実行とを並行させること。

標準省略時属性(standard default): 択一属性またはオプションが指定されておらず,該当するDEFAULTステートメントもない場合にとられる属性またはオプション。

標準ファイル(standard file): GETまたはPUTステートメントの中にFILEまたはSTRINGオプションが指定されていない場合にプロセッサーが選定するファイル。標準入力ファイルはSYSINであり,標準出力ファイルはSYSPRINTである。

ファイル(file): プログラムの中でデータ・セットを表わす名称的表現。ファイルは,開かれるたびに,ある1つのデータ・セットと結び付けられる。

ファイル属性(file attribute): ファイルの性質を記述する属性。

ファイル式(file expression): 式の値を計算した結果がファイル名になる式。

ファイル定数(file constant): ファイルに対して宣言されており,このファイルが開かれるときに完全な一組のファイル属性が存在している名前。

ファイル変数(file variable): ファイル定数を割り当てることのできる変数。この変数はFILE,VARIABLE両属性をもつものでなければならない。ファイル変数に対しては,FILE以外のファイル属性を指定することはできない。

ファイル名(file name): ファイルに対して宣言されている名前。

フィールド(データの流れの)(field(in the data stream)): データの流れの中で,1つのデータ形式項目またはスペース形式項目によって,幅が文字数単位で定義されている部分。

フィールド(ピクチャー指定の)(field(of a picture specification)): 文字ストリング・ピクチャー指定,または数字ピクチャー指定では固定小数点数について記述している部分(これが1つの数字ピクチャー指定全体であることもある)。

付加的属性(additive attributes): 省略時属性が定められておらず,しかも,必要な場合には,つねに,指定されている属性のリストに追加するか暗示しなければならない属性(言い換えれば,これらの属性が必要な場合には,これらを一組の属性に追加しなければならない)。

複合ステートメント(compound statement): 他のステートメントを含んでいるステートメント。複合ステートメントはIFとONだけである。

副次入口点(secondary entry point): ENTRYステートメントのラベル・リストの中にあるいずれかの名前によって識別される入口点。

複素数データ(complex data): 各々の項目が実部と虚部から構成されている算術データ。

含まれているテキスト(contained text): (ネストされたプロシージャーも含めて)プロシージャーの中にあるすべてのテキストから，その入口名とPROCEDUREステートメントの条件接頭語とを除いた部分。開始ブロックの中にあるすべてのテキストから，そのブロックの先頭にあるBEGINステートメントのラベルと条件接頭語とを除いた部分。内部ブロックは，外部プロシージャーに含まれている。

符号・通貨記号文字(sign and currency symbol characters): ピクチャー指定用文字s，+，-，$。次のような方法でこれらの文字を使用することができる。

1. 固定文字として使用する。この場合，1つのピクチャー指定ではこれらの文字を1回しか指定することができず，これらの文字は，対応するデータ項目の指定されている位置に現われる。
2. 浮動文字として使用する。この場合，これらの文字は(ピクチャー指定の中のストリングとして)2回以上指定される。しかし，対応するデータ項目の中では，そのデータ項目の有効数字部分の直ぐ左側に1回現われるだけである(現われない場合もある)。

浮動小数点定数(floating-point constant): 算術定数を参照のこと。

浮動文字(drifting-characters): 符号・通貨記号文字を参照のこと。

部分的に修飾された名前(partially-qualified name): 不完全な形に修飾された名前。すなわち，参照しようとしている構造のメンバーの名前，および，構造の組立ての上から見てこのメンバーの上部に連なる名前を1つ以上(しかし，すべてではない)含んでいる，修飾された名前。

不明確な参照(ambiguous reference): 参照の時点において知られている名前を唯一無二のものとして識別するのに充分なだけ修飾されていない参照。

プリプロセッサー(preprocessor): 原始プログラムの中にあるプリプロセッサー・ステートメントを調べるためのプログラム。このステートメントが見付かると，これが実行され，その結果，原始プログラムが変更される。

プリプロセッサー・ステートメント(preprocessor statement): 原始プログラムの中にあり，原始プログラムをどのように変更すべきかを指定する特殊なステートメント。このステートメントは，プリプロセッサーと出会ったとき，実行される。

プログラム(program): 1つ以上の外部プロシージャーから構成された1つの集りで，この内のいずれかのPROCEDUREステートメントにはOPTIONS(MAIN)オプションがなければならない。

プログラム制御データ(program control data): PL/Iプログラムの実行に影響を与えるためにこのプログラムの中で使用されるデータ。プログラム制御データには，入口，タスク，ファイル，ラベル，事象，ポインター，オフセット，エリアの各タイプがある。

プロシージャー(procedure): PROCEDUREステートメントに始まりENDステートメントで終るステートメントの集りであり，プログラムの一部を成し，名前の有効範囲を区切り，所属するいずれか1つの入口名を参照することによって活用化される。

プロシージャー参照(procedure reference)入口定数あるいは入口変数，あるいは後に1つ以上の引数リストをつけた組込み関数名(引数リストなしも可)。プロシージャー参照は，CALLステートメントないしCALLオプションの中か，関数参数として現われる。

プロセッサー(processor): 原始プログラム・テキスト(恐らく，プリプロセッサーによる処理の済んだテキスト)の実行の準備をするためのプログラム。

ブロック(block): 開始ブロックまたはプロシージャー・ブロック。

ブロック先導ステートメント(block heading statement): 1ブロックのステートメントの先頭に置かれるPROCEDUREステートメントまたはBEGINステートメント。

プロローグ(prologue): ブロックの活用化をするときに自動的に行なわれる操作。

文脈宣言(contextual declaration): 明示的に宣言されていない識別名が，この識別名と特定の属性との結び付きを可能にするような文脈に現われること。

分離符号(separator): 区切り符号を参照のこと。

閉止(ファイルの)(closing(of a file)): ファイルをデータ・セットから切り離すこと。

変換(conversion): 与えられた一組の属性に合致するように，値の表現をある形式から他の形式に移し替えること。

編集指示転送(edit-directed transmission): ストリーム式転送の1つのタイプで，文字の連続している流れとしてデータが現われ，関連するデータ・リストに必要な編集様式を指定する形式リストが必要であるもの。

変数(variable): データを参照するのに使用され，値を割り当てることができ，名前が与えられているもの。変数の属性は不変であるが，時点が異なれば異なる値を参照することができる。どのようなデータ・タイプに関しても，変数は要素変数，配列変数，構造変数の3つに類別される。変数は，添字付きでも修飾されていてもよいし，その両方であってもよい。また，ロケーターで修飾されていてもよい。

ポインター変数（pointer variable）： POINTER属性をもつロケーター変数。この変数の値は主記憶域内の絶対アドレスを表わす。

見かけ引数（dummy argument）： 名前によって渡すことのできない引数の値を保管するために自動的に作成される一時記憶域。

明示宣言（explicit declaration）： DECLAREステートメントの中に識別名が現われること，ラベル接頭語として識別名が現われること，またはパラメーター・リストの中に識別名が現われること。

モード（算術データの）（mode(of arithmetic data))： 算術データの特性の1つ。実数モードと複素数モードの2つがある。

多重宣言（multiple declaration）： 同じブロックの内部にある同じ識別名（異なる修飾のない）に関する2つ以上の宣言，または，同じプログラムの中にある同じ識別名（異なる属性を指定している）に関する2つ以上の外部宣言。

目標変数（target variable）： 値が割り当てられる変数。

文字（character）： 文字セットの1つの要素。

文字ストリング（character string）： ゼロまたは1つ以上の文字からなるストリング。

文字ストリング・ピクチャー・データ（character-string picture data）： 少なくとも1つのAまたはXピクチャー指定文字を必ず含むピクチャー指定によって記述されるデータ。

文字セット（character set）： 定義された文字集合体。（言語文字セットおよびデータ文字セットを参照のこと。）

問題データ（problem data）： PL／Iプログラムによって処理されるストリング・データないし算術データ。

有効な割振り（significant allocation）： エリア内の未解放の割振り，およびエリアの先頭とこの先頭から最も離れている未解放割振りの終りとの間にある，解放されているすべての割振り。後に同じ大きさの割振りが同じ位置で行なわれた場合には，最初の割振りは有効でなくなる。

有効範囲（条件接頭語の）（scope(of a condition prefix))： あるプログラムの中で，ある条件接頭語が適用される部分。

有効範囲（宣言の）（scope(of a declaration))： あるプログラムの中である特定の宣言がある特定の名前の属性指定として効力を有している部分。

有効範囲（名前の）（scope(of a name))： あるプログラムの中である特定の名前の意味が変わらない部分。

要素（element）： 1項目のデータ。いくつかのデータ項目を集めたもの（たとえば，配列）と対比される。スカラー項目。

要素式（element expressions）： 式の値を計算した結果が要素値になるような式。

要素変数（element variable）： ある1つの要素を表わしている名前。スカラー変数。

呼び出されたプロシージャー（invoked procedure）： いずれか1つの入口点から活用化されたプロシージャー。

呼出し（call）： CALLステートメントまたはCALLオプションによるサブルーチンの呼出し。

呼出し（invocation）： プロシージャーの活用化。

呼出し点（point of invocation）： 呼び出されるプロシージャーへのプロシージャー参照が呼出しブロックの中にあらわれた地点。

呼出しブロック（invoking block）： プロシージャーを活用化するためのステートメントを含んでいるブロック。

呼び出す（call）： CALLステートメントまたはCALLオプションによってサブルーチンを呼び出すこと。

呼び出す（invoke）： プロシージャーをそのいずれか1つの入口点から活用化すること。

ラベル（label）： PROCEDURE，ENTRY両ステートメント以外のステートメントを識別するのに使用される名前。ステートメント・ラベル。

ラベル定数（label constant）： PROCEDURE，ENTRY両ステートメント以外のステートメントの接頭語として使用されている（添字のつかない）名前。

ラベル接頭語（label prefix）： ステートメントの接頭語として使用されているラベル。

ラベル変数（label variable）： LABEL属性をもつものとして宣言されており，したがって，値としてラベル定数をとることのできる変数。

ラベル・リスト（ステートメントの）（label list(of a statement))： ある1つのステートメントについているすべてのラベル接頭語。

ラベル・リスト（ラベル変数の宣言の）（label list(of a label variable declaration))： キーワードLABELの直ぐ後にある，1つ以上のステートメント・ラベル定数を括弧で囲んだリストで，ここで宣言されている変数がとる値の範囲を指定するためのもの。リスト内の名前はコンマで区切られている。ラベル配列に対して指定されている場合は，このリストは，この配列の個々の要素がリスト内の任意の値をとり得ることを表わす。

リスト指示転送（list-directed transmission）： ストリーム式転送の1つのタイプで，流れの中のデータがブランクないしコンマで区切られた定数として現われ，自動的に形式指定が与えられるもの。

レコード（record）： レコード式入出力操作における転送の論理単位。

レベル番号（level number）： DECLAREないしALLOCATEステートメントの中にある，構造の階層における名前の位置を指定するための無符号の10進整数定数。レベル番号はこの番号が適用される名前の直前にあり，レベル番号とこの名前とは名前の区切り符号によって区切られる。ENTRY属性の指定のパラメーター記述子においては，レベル番号は名前を伴なわない。

レベル１の変数（level-one variable）： 大構造名。構造に含まれていない添字の付かない変数。

連結（concatenation）： 指定されている順序に２つのストリングを結び付け，２つのストリングの長さの和に等しい長さをもつ１つのストリングにする操作。演算子｜｜によってこの操作を指定する。

連続参照（connected reference）： 連続している記憶域への参照。プログラムの実行以前に，この記憶域が連続していることが明白でなければならない。

連続していない記憶域（non-connected storage）： ある１つの名前によって参照することのできる関連するいくつかのデータ項目を記憶しているが，この名前によって参照されない他のデータ項目によって分断されているような，記憶域内の散在している場所。構造の配列の中の添字付でない基本項目名によって参照されている記憶域や，配列の断面を参照している添字付の名前（任意の要素式の左側に星印のある添字リストをもつ）によって参照される記憶域などがこの例である。

連続している記憶域（connected storage）： １つの名前によって参照することのできる，途切れることなく線型につながっている諸項目の主記憶域。

ロケーター修飾（locator-qualification）： 基底付変数への参照において，ここで参照される基底付変数の世代を指定するために基底付変数の左側に矢印でつながれたロケーター変数または関数参照，または，ロケーター変数と基底付参照との暗示的な連結。

ロケーター変数（locator variable）： 主記憶域における ある変数または緩衝域の位置を識別するための値をもつ変数。

論理演算子（logical operators）： ビット・ストリング演算子，¬（not），&（and），|（or）。

論理レベル（構造のメンバーの）（logical level(of a structure member)）： すべてのレベル番号を逐次並べたとき，すなわち，隣接するレベル番号の差が１になるようすべてのレベル番号を並べたときの，レベル番号の示す深さ。

割振り（allocation）：

1. ある変数のために主記憶域を確保すること。
2. 割り振られている変数の１つの世代。

割り振られている変数（allocated variable）： 主記憶域と結びつけられており，その主記憶域が未だ解放されていない変数。

割込み（interrupt）： 解禁状態にあるＯＮ条件が引き起されるか，あるいは警告の結果として（恐らくは一時的に）プログラムの制御の流れを変えること。

割当て（assignment）： 変数に値を与えるための操作。

ＤＯグループ（do-group）： ＤＯステートメントで始まり，対応するＥＮＤステートメントによって終る一連のステートメント。制御のために使用される。

ＤＯループ（do-loop）： 反復ＤＯグループを参照のこと。

ＯＮ条件（on-condition）： ＰＬ／Ｉプログラムの中で，プログラム割込みを引き起す可能性のある事柄が起こること。ＯＮ条件は，予期できないエラーの検出であることもあり，予期できるがいつ起こるか分らない事柄の検出であることもある。

ＲＥＦＥＲ式（REFER expression）： キーワードＲＥＦＥＲの直前にある式で，ＲＥＦＥＲオプションをもつ基底付変数がＡＬＬＯＣＡＴＥ ないしＬＯＣＡＴＥステートメントによって割り振られるときに最初の限界，長さ，または大きさがこの式からとられる。

ＲＥＦＥＲの目的語（REFER object）： ＲＥＦＥＲオプションの中にある，基底付構造のあるメンバーの現在の限界，長さ，または大きさを指定するための添字のついていない要素変数。これは，この構造のメンバーでなければならず，またＲＥＦＥＲオプションで宣言されているメンバーの前になければならない。

２進法（binary）： ２を基数とする記数法。

１０進数字桁文字（decimal digit character）： ９というピクチャー指定用文字。

１０進法（decimal）： １０を基数とする記数法。

１０進ピクチャー・データ（decimal picture data）： 次のタイプのピクチャー指定用文字を含むピクチャー指定用文字によって指定される算術ピクチャー・データ。

1. １０進数字桁文字。
2. 仮想小数点ピクチャー文字。
3. ゼロ消去文字。
4. 符号・通貨記号文字。
5. 挿入文字。
6. 商用文字。
7. 指数文字。

ア 行

サ　行

タ　行

ナ 行

ハ 行

ラ　行

ワ 行

英 字

数字

記　号

ご意見記入用紙

N：GC33－0005－3

ご芳名　：

DOS PL／I 最適化コンパイラー：
言 語 解 説 書

貴社名　：

所属部門：

所在地　：〒

本書をより使い易いものとするために，読者の皆様からの貴重なご意見やご提案をお待ちいたしております。ご記入に際しましては，特に次のような事柄について具体的にご指摘いただければ幸いです。

- 理解し難い部分（説明が不明確な部分）
- 邦訳に起因するわかりにくい用語
- 誤字や誤植
- その他

該当ページ	該当個所およびご意見

その他，お気付きの点についても，ご自由にご記入ください。

- この用紙は，弊社の製品に関する技術的なご質問や追加資料のご注文にはご使用にならないでください。そのようなご質問やご注文は，弊社の担当員あるいは担当営業所にお寄せくださるようお願いいたします。
- なお，お寄せいただきましたご意見やご提案は，すべて弊社の所有とさせていただき，その採用の可否，および採用する場合の方法については，弊社の判断において決定させていただきます。
- ご協力ありがとうございました。

切取り線

折込み線　折込み線

日本アイ・ビー・エム(株)　社内郵便扱い

CH-355-B 営業資料センター
資料開発担当行

担当：　営業部　営業所　課

氏名

（社内郵便番号　）

折込み線　（この面を内側にして折り込んでください。）　折込み線

● ご記入くださいましたら点線の部分で折り込み，ホッチキス等で止めたうえ，弊社の営業担当員またはシステムズ・エンジニアにお渡しください。

● このままで郵便物として投函なさらないようご注意ください。

N:GC33-0005-3

DOS PL/I 最適化コンパイラー：言語解説書 (File No. S360/S370-29) Printed in Japan N:GC33-0005-3

IBM

日本アイ・ビー・エム株式会社

東京都港区六本木 3 - 2 -12 〒106

TEL (03) 586-1111